A MODEL FOR A CONTEXT AWARE MACHINE-BASED PERSONAL MEMORY MANAGER AND ITS IMPLEMENTATION USING A VISUAL PROGRAMMING ENVIRONMENT

A thesis submitted in fulfilment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

of

RHODES UNIVERSITY

by

MELEKAM ASRAT TSEGAYE

August 2006

Abstract

Memory is a part of cognition. It is essential for an individual to function normally in society. It encompasses an individual's lifetime experience, thus defining his identity. This thesis develops the concept of a machine-based personal memory manager which captures and manages an individual's day-to-day external memories. Rather than accumulating large amounts of data which has to be mined for useful memories, the machine-based memory manager automatically organizes memories as they are captured to enable their quick retrieval and use. The main functions of the machine-based memory manager envisioned in this thesis are the support and the augmentation of an individual's biological memory system.

In the thesis, a model for a machine-based memory manager is developed. A visual programming environment, which can be used to build context aware applications as well as a proof-of-concept machine-based memory manager, is conceptualized and implemented. An experimental machine-based memory manager is implemented and evaluated.

The model describes a machine-based memory manager which manages an individual's external memories by context. It addresses the management of external memories which accumulate over long periods of time by proposing a context aware file system which automatically organizes external memories by context. It describes how personal memory management can be facilitated by machine using six entities (life streams, memory producers, memory consumers, a memory manager, memory fragments and context descriptors) and the processes in which these entities participate (memory capture, memory encoding and decoding, memory decoding and retrieval).

The visual programming environment represents a development tool which contains facilities that support context aware application programming. For example, it provides facilities which enable the definition and use of virtual sensors. It enables rapid

programming with a focus on component re-use and dynamic composition of applications through a visual interface.

The experimental machine-based memory manager serves as an example implementation of the machine-based memory manager which is described by the model developed in this thesis. The hardware used in its implementation consists of widely available components such as a camera, microphone and sub-notebook computer which are assembled in the form of a wearable computer. The software is constructed using the visual programming environment developed in this thesis. It contains multiple sensor drivers, context interpreters, a context aware file system as well as memory retrieval and presentation interfaces.

The evaluation of the machine-based memory manager shows that it is possible to create a machine which monitors the states of an individual and his environment, and manages his external memories, thus supporting and augmenting his biological memory.

Keywords

personal memory management, memory centric model, context aware application development, visual programming, wearable computing, ubiquitous computing

Acknowledgments

Firstly, I would like to thank Shaun Bangay and Alfredo Terzoli for serving as supervisors during the last three years. They have provided a distinctive but complementary style of supervision. We have had many meetings (real and virtual), resulting in interesting discussions and debates, all of which have helped shape the thesis.

I am also grateful to Aruna Manipura, Victor Watur, Emeilie Rauthenbach and Jan Richter for giving up their time to proof read the thesis, Anthony Sullivan for his advice on building the a2d device, Lorenzo Dalvit for his input at the end, the Andrew Mellon Foundation for sponsoring this work and the Dean of Research's office for facilitating the sponsorship.

Lastly, but most importantly, I would like to thank my parents for their unconditional support.

Obligatory note:
The financial assistance from the Andrew Mellon Postgraduate Scholarship towards this research is hereby acknowledged. This research was conducted using facilities made available by the Rhodes University Centre of Excellence (COE). Opinions expressed and conclusions arrived at are those of the author and are not necessarily attributed to Rhodes University or the donor.

Table of Contents

Figures

Tables

1 Introduction

We are living in an era where artificial computing devices which can capture our experiences as external memories are ubiquitous. However, these devices are fragmented and lack the ability to automatically organize memories, which accumulate during day-to-day use, in order to enable quick retrieval and use of memories. It would be ideal if a computer could act as a personal memory manager which captures a person's day-to-day external memories in the background and automatically organizes them for quick retrieval and use. Such a machine would enable an individual to have a photographic memory. Unlike the human brain, which is susceptible to forgetting, a computer can quickly retrieve and present requested external memories in the form they were captured. It can also make use of artificial sensors to capture external memories which humans do not naturally perceive. In its role as a personal memory manager, the machine would support and augment an individual's biological memory system.

Motivated by the many advantages a personal machine-based memory manager would provide, a model for a machine-based memory manager, the software framework and infrastructure needed to build a machine-based memory manager and an experimental machine-based memory manager, are developed.

1.1 Problem statement

The problem which is addressed by this thesis is the support and augmentation of an individual's biological memory system using a computer. Such a machine should enable a person to remember all their external memories, capture those which his biological system cannot, and retrieve these memories by describing the circumstances of their formation. It should also provide an effective user interface for memory retrieval.

Addressing the main problem outlined above, requires the resolution of a number of problems and sub-problems.

Problem 1: identifying existing research to support the hypothesis of this work, which states that a person can benefit from a machine-based personal memory manager

- establishing that a person has problems with personal memory management
- identifying the current state of the art in electronic memory aids

Problem 2: identifying an effective means of managing an individual's day to day memories

- establishing the requirements for using the identified solution for day to day memory management
- identifying existing solutions which use this method of memory management

Problem 3: developing a model for a machine-based personal memory manager

- developing a model which explains how a machine will capture, analyze, store and facilitate the retrieval of external memories

Problem 4: identifying or developing a software infrastructure which can be used to construct an experimental machine-based personal memory manager

- developing a set of requirements for the software infrastructure
- if an existing software infrastructure is not available, developing a new one

Problem 5: developing a proof of concept of the model

- implementing an experimental machine-based memory manager which is based on the proposed model
- Validation of the correct functioning of the machine-based memory manager.

1.2 Research methodology

This research work was conducted over a period of three years (2003-2005). During this period, proposals which were made and methods which were developed by other researchers to address the problems identified in section 1.1 were considered. Relevant models of the human memory system as well as an effective solution for human memory management were identified and used as the basis for the construction of a model which can be used to develop a machine-based personal memory manager. In order to test the proposed model practically, a proof of concept implementation of a machine-based personal memory management system based on the developed model would normally be implemented. A survey of related research showed that researchers often construct experimental systems which emulate a small subset of the facilities offered by the human memory subsystem. This is a conventional, tried and tested approach which should be followed when the interest of the researcher is in a small area of the human memory system. However, as this work is interested in addressing the larger problem of machine-based personal memory management, it took an alternative approach. A software infrastructure which enables experimentation with a larger more complex system was conceptualized and implemented. With the aid of this software infrastructure, a machine-based personal memory manager was implemented and evaluated. Whenever possible, rather than implementing various subsystems of the machine-based memory manager from scratch, existing implementations of subsystems which have been developed by other individuals were used.

In summary, the following steps were taken:

1. A survey and comparison of relevant literature was conducted to establish that the identified problems exist and to consider previous solutions if they existed
2. A model that incorporates concepts from existing solutions and which is to be used to solve the main problem was developed

3. A software infrastructure to support the development of an experimental system, which incorporates existing solutions and can be used to test the proposed model was developed
4. An experimental system which is based on the model was constructed with the aid of the software infrastructure and evaluated

1.3 Thesis contributions

A number of contributions are made by this thesis:

Contribution 1: development of a model for a machine-based personal memory manager

The model adopts the information processing model of memory and context based memory management to explain how a machine-based memory manager can capture, store and facilitate the retrieval of an individual's day to day external memories using a few clearly defined entities and processes. The model also explains how external memories which accumulate over long periods of time (such as the lifetime of an individual) can be managed in order to enable fast and effective memory retrieval. The model represents an approach which can be used to address the problem of support and augmentation of the human biological memory system with the help of a machine.

Contribution 2: development of the concept of a software environment for prototyping context aware systems and providing a proof of concept implementation of such a system

The software environment represents an integrated development environment with a visual programming interface, which provides generic support for the development of context aware applications. The machine-based memory manager developed using this environment demonstrates the creation of a complex, context aware system that makes use of multiple sensors, context interpreters and human-computer interaction interfaces. The platform simplifies the creation of experimental context aware systems from reusable components on general purpose computers. It also provides facilities which encourage

less dependence on hardware, such as the use of virtual sensors, when initially prototyping applications.

Contribution 3: implementation of an experimental machine-based personal memory manager

An experimental machine-based personal memory manager, which is based on the model developed in this thesis, is implemented and evaluated. It is packaged as a wearable computer which can accompany an individual everywhere. It consists of multiple sensors, automatic context interpreters, manual context interpreters, an implementation of a context aware file system (for lifelong context based personal memory storage) and multiple memory presentation interfaces. It enables memory tagging and memory retrieval using speech and remote control based interfaces. It enables individuals to retrieve their memories using multiple context descriptors. The experimental platform represents a prototype which can be used to conduct practical research into the support and augmentation of the human biological memory system.

1.4 Structure of the thesis

The material presented in the thesis is divided into eight chapters. Chapter 2 motivates for the idea that a person's memories should be enhanced with a machine-based personal memory manager. It highlights the problem of personal memory management and identifies proposed solutions from existing research. Chapter 3 explores the concept of context awareness as developed by researchers in context aware computing. Context awareness is identified as an effective solution to machine-based personal memory management. Chapter 4 develops a model for a machine-based memory manager which is based on concepts from the research considered in chapter 2 (psychological models of memory) and chapter 3 (context aware computing). Chapter 5 introduces the concept of a software environment which can be used for prototyping context aware applications. Requirements of the software environment are identified. The implementation of the software environment is presented. Chapter 6 presents the implementation of an

experimental machine-based memory manager which is based on the model introduced in chapter 4. It shows how the machine-based memory manager is built using the software environment developed in chapter 5. Chapter 7 presents the evaluation of the machine-based memory manager through a series of experiments. Chapter 8 presents the conclusion of this work.

2 Background

After millions of years of evolution the human brain has developed abilities such as memory, learning, reasoning and imagination which enable humans to adapt and survive in their natural environment. Not being satisfied with these natural abilities, humans have constantly sought to improve them artificially by developing devices which augment these abilities as well as methods of using these devices creatively. Of all the devices invented by people, the general purpose computer is a good example of a device which shows how much humans have progressed technologically. The computer has enabled individuals to attain "super human" abilities, as it can process large amounts of data at high speeds and promises to be the base of technology which will enable individuals to attain photographic memory. Humans have also developed the Internet, an infrastructure consisting of millions of connected computers which enables them to be connected constantly to billions of other humans, massively increasing the amount of external memory available to the individual. In order to help individuals keep track of their external memories, this thesis proposes that a machine-based personal memory manager is necessary.

Before attempting to develop a machine-based memory manager it is necessary to consider how the human memory system works. It is also important to consider existing machine-based memory aids and proposals which are made by researchers for future machine-based memory aids, in order to identify what is lacking in existing solutions and proposals. This chapter focuses on the above issues. Models developed by psychologists explaining how the human memory system works are examined and weaknesses which exist within it are identified. Parts of the human memory system that can benefit from a machine-based memory aid are identified. Existing proposals for machine-based memory aids are examined with a focus on problems that exist in current personal memory management solutions. Software and hardware platforms which are part of existing machine-based memory aid solutions are considered and potential platforms for future

machine-based memory aids are identified. A comparison of existing research work is performed and areas that this work can make contributions in are identified.

2.1 Human memory

Human memory is defined by Anderson [Anderson, 2000] as a record of a person's experiences and knowledge. Most people take for granted their brain's ability to learn from experience and allow them to recall past events instantly. The importance of a fully functional memory subsystem can only be appreciated by studying the difficulties faced by individuals who have medical problems with their memory. The social impact of the loss of memory on people who develop medical conditions which lead to the deterioration of memory is well known. Alzheimer's disease is an example where the loss of memory is so severe that some individuals cannot even recognise their own family members. An inability to remember one's history can lead to the loss of self identity. An inability to form new memories can lead to social alienation and isolation. Memory loss, however, is a condition which affects all people and not just those with medical conditions.

2.1.1 Theories of human memory

Various theories have been proposed in the past to explain how human memory works. Herrmann [Herrmann *et al*, 1992] proposes a taxonomy of general memory theories which consists of the following categories:

1. Biological: theories which focus on the biological bases of memory in terms of brain chemistry
2. Classical: the oldest theory of memory, explains memory in terms of mnemonics which are previously learned procedures which help learn new information
3. Behavioural: theories which explain memory as a response conditioned by a stimulus
4. Motivational: theories which explain memory in terms of the causes of forgetting, for example, inability to deal with anxiety

5. Educational: theories which assume memory is associative in nature and formation of new memory is dependent on associations previously learnt by individuals
6. Information processing: theories which explain memory in terms of information which is acquired, processed and stored in a way it can be retrieved later under the control of a central processor. Based on previous experiences, the central processor controls how much attention is given by an individual to a particular task
7. Cognitive: theories which are based on information processing theories of memory but assert that information processing depends on the type of task and domains of information encountered by individuals and their experience with the type of task
8. Multimodal: rather than focusing on how memory is processed by various subsystems and identification of these subsystems, multimodal memory theory concentrates on how memory can be improved by considering physiological and psychological conditions

The existence of all these theories of memory shows that the human memory system is complex. The information processing and multimodal theories of memory are important to consider in the context of developing a machine-based memory manager because they provide models which are relevant to a machine-based memory aid implementation. This section considers these two theories in detail. Their contributions to a model for a machine-based memory manager are later summarized in section 2.1.4.

2.1.1.1 The multi-store model of memory

An influential theory of memory which forms the base of modern memory theories is the multi-store model of memory which is shown in Figure 1. It was first proposed by Atkinson and Shiffrin [Atkinson *et al*, 1968] as a theory of human memory. The model consists of three distinct forms of memory: sensory memory, short-term memory (STM) and long-term memory (LTM). Information perceived by one or more of the senses is first stored in sensory memory for brief periods of time where it can go through stages of processing, after which it is passed on to STM memory which is also a temporary storage for memories. Through the process of rehearsal, information in STM can be committed to LTM to be retained for longer periods of time. According to this model, the human

memory system consists of a chain of these three different types of memory systems though which information flows, as shown in Figure 1.

Long-term memory is further divided into two types: procedural (memories which cannot be consciously accessed or described) and declarative (memories which can be consciously accessed/described). As an example of procedural memory Parkin [Parkin, 1999] cites acquired skills such as knowing how to ride a bicycle or speaking a native language, which he says are difficult to describe or directly transfer to others. Declarative memory is consciously accessible memory which can be further divided into two types of memories: semantic and episodic. Parkin [Parkin, 1997] defines semantic memory as a person's general knowledge of concepts, rules and languages which make up the world and states that the use of semantic memory is not dependent on the time or event at which the particular semantic memory is formed. Episodic memory is described by Tulving [Tulving, 1985] as a system which receives and stores information about temporally dated events and is a record of a person's life. The various types of memory presented thus serve as useful categorizations for understanding how human memory works but do not mean that human memory is made up of separate memory systems. Parkin [Parkin, 1997] and Baddeley [Baddeley, 1999] assert that an individual's behaviour is governed by a combination of the different types of memory described.

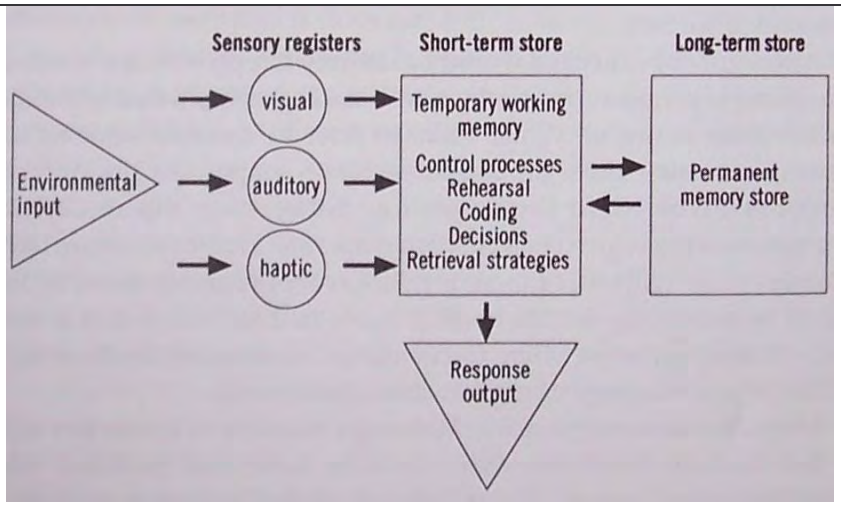


Figure 1. Atkinson's multi-store model of memory as illustrated by Baddeley [Baddeley, 1999]

2.1.1.2 Developments after the multi store model of memory

In the early 1970s it was found that the multi-store model of memory proposed by Atkinson [Atkinson *et al*, 1968], discussed in section 2.1.1.1, has shortcomings. It was assumed that short-term memory is necessary for learning new material but experiments conducted by psychologists with patients who have a defective short-term memory showed that their long-term memory worked without problems [Baddeley, 1999]. The levels-of-processing model of memory [Craik *et al*, 1972] was proposed to address shortcomings of Atkinson's model of memory. In this model, processing of information is controlled by a central processor, called the executive. A new stimulus is analyzed through a series of steps, each representing a different dimension of stimulus. The levels-of-processing model can be illustrated with the following example [Parkin, 1997]. For vocal memory, words are processed by the central processor at three levels: orthographic (spelling of words), phonological (sounds of words) and semantic (meaning of words), with semantic interpretation representing the deepest level of processing.

A third model of memory proposed by Baddeley [Baddeley *et al*, 1974], after the levels-of-processing model of memory, is known as the working memory model. The working memory model is created to explain memory organization for supporting everyday activities. It replaces the single short-term memory subsystem in Atkinson's model with three new subsystems. The working memory model is shown in Figure 2 and consists of a central executive which controls two subsystems called a visuo-spatial sketchpad and phonological loop. The central executive has the ability to store or retrieve memory from these subsystems [Morris *et al,* 1994], [Anderson, 2000]. It also has regulatory functions such as the control of attention. The visuo-spatial sketchpad is a subsystem which can process and store visual and spatial information and is essential for learning new visual information such as faces, objects and spatial layouts of objects. The phonological loop is another subsystem used for processing verbal information. It is responsible for phenomenon known as the "inner voice" and "inner ear" which enables people to rehearse verbal information without making sounds as well as listen to them.

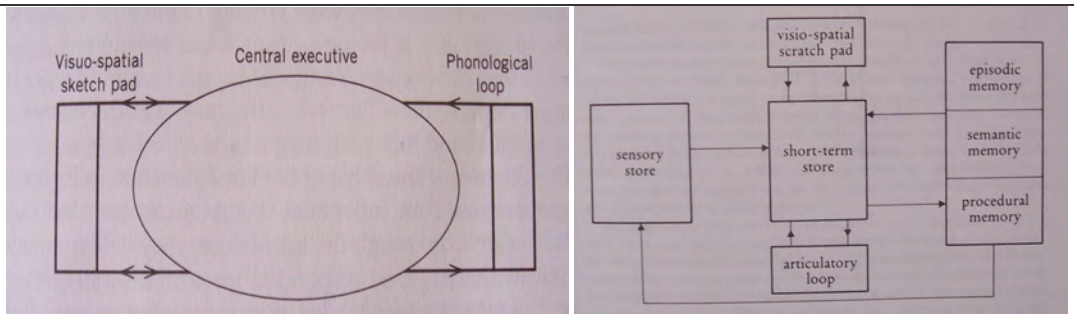


Figure 2. (1) Working memory model of human memory as illustrated by Anderson [Anderson, 2000]. (2)Working memory and the multi-store model of memory as illustrated by Parkin [Parkin, 1997]

2.1.1.3 Multimodal theory of memory

Multimodal theory of memory examines memory from a different angle: memory improvement by optimizing physiological and psychological systems in which a memory system is embedded. Herrmann [Herrmann *et al*, 1992] points out that memory has a psychological mode of operation which involves a person's physiological condition, emotional state, attitude, social behaviour, sensory and perceptual interaction with the environment and these affect memory, and cannot be solely explained by theories such as those presented in section 2.1.1.1 and 2.1.1.2. These theories, Herrmann argues, consider a memory system only in terms of the different types of memory, processing performed and the mechanisms which process memory but ignore important physiological and psychological factors which influence it.

2.1.2 Aspects of human memory

The human memory system is a biological system made up of cells (neurons) which communicate and retain information through electro-chemical processes. The system is prone to external influences such as diseases, drugs and social situations which can disrupt the normal operation of the nervous system and impact on the individuals' survival in their habitat. For example, storage failure is a phenomenon demonstrated by administering certain drugs to patients which are known to interfere in memory formation. Parkin [Parkin, 1997] describes that while the drugs are active patients are unable to form new memories.

In a healthy memory system, information from the environment is acquired by the human sensory system and passed to the human brain which encodes and stores the acquired information either for temporary or long-term storage. The encoding process is not always accurate as illustrated by false memory phenomenon whereby individuals interpret and form memories of events incorrectly [Baddeley, 1999]. During information acquisition sensors can be impeded from picking up interesting stimuli by unwanted stimuli (noise). During encoding, existing or new memories can interfere with the encoding processes. Existing memories can prevent the creation of new memories and new memories can displace old memories. Anderson [Anderson, 2003] describes that interference has a disruptive effect on memory and is thought to be a cause of forgetting. Forgetting can also occur if information held in short-term memory is discarded when no longer needed.

When required, memories can be retrieved through the process of remembering which involves the human brain having to decode and retrieve stored memories. If memories exist in storage but cannot be retrieved, a retrieval failure is said to have occurred. However, Parkin [Parkin, 1999] states that such memories can resurface at a later time unexpectedly, in a process known as spontaneous recovery.

2.1.3 Weaknesses in the human memory system

The discussion in the previous sections has identified some aspects of the human memory system which can be seen as weaknesses, such as:

- deliberately committing memories to long-term storage is not a voluntary process in humans

A great deal of effort such as repetition and practice is necessary for humans to commit memories to long-term storage.

- forgetting, which is not strictly a weakness

Forgetting is identified in the discussion of aspects of human memory and in section 2.1.2. Forgetting is not strictly a weakness because it has benefits in some cases, for example, it can help humans emotionally and socially.

- weaknesses exist in the information acquisition, memory encoding, storage and retrieval subsystems

The human memory system is a biological system. It also has a limited set of sensors for monitoring the environment. The system is prone to malfunction if there are electrochemical imbalances in the system or if it is attacked by foreign organisms which cause disease. As a result, the memory encoding subsystem can encode the wrong information, can fail to store encoded information or can fail to retrieve information which already exists in memory.

- information retained in memory is fully accessible to the individual only. It is difficult to accurately transfer information from person to person

Humans exchange information stored in their memory by providing verbal, auditory and visual descriptions. This form of communication by humans can be inefficient and inaccurate. Human to human communication is a limited way to exchange memories. It is slow and cannot, for example, be used to accurately transfer images and sounds from individual to individual.

2.1.4 The contributions of psychological models of memory

In proposing models which attempt to explain how the human memory system works, psychologists have identified a number of subsystems which are important abstractions of the human memory system. These include:

- the senses

Sensors enable the human memory system to acquire information from its environment which forms the basis of memory. Therefore they are a vital part of the human memory

system. A machine-based memory aid needs to have sensors that are similar to or better than sensors which are part of the human memory system. The former enables a machine to capture the same information as the human memory system. The latter enables a machine to augment the human memory system.

- short-term memory

Short-term memory is important in processing new memories so that they can be used to aid cognitive tasks and/or can be committed to long-term storage. In the working model of memory, two short-term memory subsystems are identified:

1. *The visuo-spatial sketchpad*: a machine-based memory manager can augment the ability that humans have, to seamlessly process visual and spatial information by enabling them to capture, retrieve and work with visual and spatial memories using artificial sensors and sensor data analysis components. In addition, the machine-based memory manager needs to provide a means of seamlessly transferring memories between it and the visuo-spatial sketchpad.
2. *The phonological loop*: a machine-based memory aid can provide facilities that support and augment the phonological loop. For example, it can eliminate the need for people to learn languages by providing a facility such as a universal language translator which augments the phonological loop. In addition, a machine-based memory manager needs to provide a means of seamlessly transferring memories between it and the phonological loop.

- long-term memory (procedural and declarative)

A machine-based memory manager can enable individuals never to forget their memories, whether they are procedural or declarative and enable them to exchange them with other humans.

Psychological models also identify processes which are part of human memory management, such as information acquisition from the environment (memory capture) and their subsequent storage (memory encoding and storage). The working model of

memory is important because it identifies subsystems in short-term memory that can be supported and augmented by a machine. The levels-of-processing model of memory is important as it provides an approach for a machine-based memory manager to analyze sensor data. The multimodal model of memory is important as it proposes that a machine-based memory manager needs to consider providing different modes of operation based on a person's psychological and physiological conditions.

2.2 A survey of machine-based memory aids

Weiser [Weiser, 1991, 1994], Norman [Norman, 1998] and Raskin [Raskin, 2005] advocate for the creation of machines which are transparent, serve as appliances and help manage a person's content (memories). Transparent systems stay in the background, monitor what people are doing automatically and provide useful facilities without interfering with a person's day-to-day activities. Machine-based memory aids are a subset of such machines. This section considers current personal memory management systems and the problems that arise when individuals make use of them. Research systems that go beyond current systems and potential platforms for future machine-based memory aids are also considered.

2.2.1 Current personal memory management systems and related problems

Many forms of personal memory management systems are currently in use. Provided facilities include manual memory capture, memory storage and simple forms of memory retrieval (retrieval with the help of metadata and text annotation). Examples of current personal memory management systems include personal information management (PIM) systems such as PDAs, talking alarm clocks, voice recorders and wrist watches [Koblentz, 2005]. Features common to most PIM systems include address book, schedule and communication (e.g. email) management. Recently proposed research systems such as Haystack [Karger *et al*, 2005] and OSAF [OSAF Chandler, 2005] also provide the same type of memory management as PIMs. A decade ago, Lamming [Lamming *et al*, 1994] observed that personal memory management systems in his time were only high

technology versions of conventional dairies, notebooks and alarm clocks. This observation still applies to current personal memory management systems.

There are number of problems related to the use of current personal memory management systems. These include system heterogeneity, the scope of facilities provided and user interface issues.

2.2.1.1 System heterogeneity

The existence of different types of personal memory management systems, appliances and protocols which people can use, means that it is difficult for memories of individuals to move from one device to another transparently. Even though standards for data exchange and synchronization such as WebDav [IETF, 1999b], UPnP [UPnP, 2005] and SyncML [OMA, 2005] exist, not every system uses them. Often, individuals have to convert and move their memories from device to device manually.

2.2.1.2 Scope of facilities provided

Some personal memory management systems provide many facilities on a single device, which do not directly support an individual's cognitive activities. Examples are the current generation of mobile devices such as cellular phones running Symbian OS [Symbian, 2005] and PDAs running the Pocket PC OS [Microsoft, 2005d] or Palm OS [Palm, 2005]. These devices attempt to provide many facilities similar to those found on desktop computers. Norman [Norman, 1998] presents convincing arguments why this is a bad idea. One of his arguments is that integrating more functionality in a single device means increasing its complexity. He states that appliances that are good at what they do are more useful to people than single devices which provide many facilities.

2.2.1.3 User interface problems

Dunlop [Dunlop *et al*, 2002] and York [York *et al*, 2004] point out that current personal memory management systems such as PIMs attempt to offer a multitude of facilities through interfaces which are not convenient for users to interact with. As a result, individuals are forced to use a PIM device out of necessity rather than convenience. This impacts on users, as they are forced to give too much attention to devices rather than

fully concentrating on accomplishing their tasks. On mobile systems, such as PDAs and cellular phones, designers have developed user interfaces which are supposedly appropriate for use when individuals are mobile. These devices contain smaller versions of keyboards, graphical displays and in certain cases, a stylus, which is used for pen based interaction. While graphic displays enable communication using pictures which are effortless for people to process (Hoffman [Hoffman, 1998] provides convincing arguments for this), the main form of input still remains text based which is not the best way for people to express themselves. Zhai [Zhai *et al*, 2005] puts text based input devices which are currently in use into six categories (QWERTY keyboard, speech recognition, handwriting recognition, mobile keyboards, gestures and stylus keyboards) and rates speech and handwriting recognizers as still being too slow for facilitating natural interaction with devices.

Baber [Baber *et al*, 2002] argues that user interfaces should enable people to use interaction skills which they have acquired throughout their lives rather than require them to learn new sets of commands and interaction techniques with every new device. For example, Bartneck [Bartneck, 2001] and Sebe [Sebe *et al*, 2005] observe that expression of emotions is a natural act to people and therefore the use of emotions should be considered when creating user interfaces but realises that emotions are difficult to describe. Bartneck asserts that devices should not only recognize emotions, but they should also be able to express them.

2.2.2 Proposals for machine-based memory aids that go beyond current personal memory managers

A number of proposals have been made for the creation of machine-based memory aids. The proposals include Lamming's memory prosthesis [Lamming *et al*, 1994], Mitchell's learning apprentice [Mitchell *et al*, 1994], DeVaul's memory glasses [DeVaul *et al*, 2000], Rhodes's remembrance agent [Rhodes *et al*, 1996, 1997b], Dey's Cybreminder [Dey *et al*, 2000b], Mann's [Mann, 1998b, 2001] personal imaging and Kawamura's framework for memory augmentation [Kawamura *et al*, 2002, 2004].

There is a general consensus amongst the various authors on the need for a machine-based memory aid that acts as a memory appliance which can capture, store and facilitate the retrieval of a person's day-to-day memories. DeVaul, Dey, Lamming and Rhodes are particularly interested in a machine-based memory manager that assists individuals to remember retrospective and prospective memories. In addition to supporting a person's biological memory system, DeVaul, Mann and Kawamura recognize that a machine-based memory manager can also augment their biological memory system. DeVaul develops the idea of the memory glasses that presents visual memories to users through a small display which is placed in front of the human eye. Mann conceptualises a personal imaging device which is similar to the memory glasses but can also modify visual memories before they enter an individual's eyes. Kawamura conceptualizes memory retrieval through interactions with real world objects. DeVaul, Lamming, Mann and Rhodes realise that an intimate computer that follows people wherever they go would be necessary to facilitate the functions of a machine-based memory manager.

In order to manage captured memories Dey, DeVaul, Lamming, Kawamura and Rhodes realise that the use of environmental information in tagging and facilitating the retrieval of a personal memories would be necessary. When presenting retrieved memories to individuals, a machine-based personal memory manager should be able to present them in a way that does not overwhelm its user [Lamming *et al*, 1994]. A personal memory manager should be able to learn a user's habits and provide facilities based on what has been learnt, for example, suggesting the duration of meetings based of previously observed length of meetings [Mitchell *et al*, 1994].

2.2.3 Lifelogs - lifelong personal memory management

A lifelog refers to a record of memories which span long periods of time kept by an individual. Some people keep lifelogs (a daily diary) mostly using text to describe their everyday experiences. The notion of keeping a lifelog in electronic form has been proposed as far back as 1945 [Bush, 1945]. Electronic encoding and storage of memories has the potential to enable individuals to access their memories effortlessly and instantly. Thus Vannevar Bush [Bush, 1945] proposed the Memex, a device with these properties.

Simpson [Simpson *et al*, 1996] on the 50th anniversary of the Memex, describes that Bush envisioned the Memex as device (rather than a system of devices) which helps people record, organize, search and exchange the large amount of information present in their lives. The Memex resembles the modern Internet connected computer, since it serves as a gateway to vast amounts of information. The Internet and personal computer use have become ubiquitous, but fully automatic and transparent management of a person's lifelong personal memories with the help of machines has still not been achieved.

2.2.3.1 Current lifelog systems

A wide variety of applications which enable individuals to keep lifelogs are currently available on mobile and desktop computers as well as the Internet. No single piece of software which provides the required functionality for lifelogging (memory capture, memory encoding and flexible memory retrieval) is currently available. Users have to use a combination of tools (software and hardware) in order to perform lifelogging. Memory capture and encoding is performed with the help of a variety of PIM devices which encode memories into electronic documents. To help in retrieval, memory items are annotated with text or descriptive metadata. Examples of desktop based lifelogging software include Nokia's LifeBlog [Nokia, 2005] which, in combination with a camera phone, enables individuals to keep video/audio lifelogs. Recently desktop search engines have also emerged as potential tools for supporting lifelogging.

Desktop search engines

Search engines enable a convenient interface for lifelogging. Individuals do not have to worry about organizing their data on their computer. Instead they search for items in memory when they need them. The task of locating and indexing documents is left to a search engine's document indexer. The use of desktop search engines has recently gained in popularity. Although a file search facility has always been available on desktop computers, dedicated applications which enable users to find documents on their desktop computers without the added burden of installing a local web search engine have only recently appeared. Examples are Google Desktop Search [Google, 2005], Copernic Desktop Search [Copernic, 2005], MSN Desktop Search [Microsoft, 2005] and Blink X

Desktop Search [Blinkx, 2005]. Desktop search engines such as Blink X claim to provide extra features to users such as implementation of the virtual folder concept. A virtual folder is a desktop folder which represents a saved query that can be executed each time a user opens the virtual folder. Thus the contents of a folder are dynamic and depend on the results that match the query which is associated to a particular virtual folder.

A search based desktop user interface is being developed in most modern operating systems, such as Microsoft's next operating system [Microsoft, 2005g] which is to appear in late 2006 and also in later versions of the KDE window system [KDE, 2005] in Linux. Early implementations of search based desktop interfaces are already present in Apple Corporation's OSX operating system [Apple, 2005] as well as the Gnome window system [Beagle, 2005] in Linux. Applications which search and index documents on a desktop computer have been available even before the introduction of the desktop search engines described above. In the UNIX environment, slocate [Gnu, 2005] is a tool which helps users locate files by matching against their file names. Glimpse Index [Glimpse, 2002] and mnoGoSearch [mnoGoSearch, 2005] are search engines which index text documents by their content and make them searchable to users. What makes the current set of desktop search engines different is that they index a variety of electronic document types in addition to text documents and enable the use of new desktop user interface features such as virtual folders. Their ultimate aim is to simplify desktop document management by creating an environment which enables users to *search and find, rather than organize*.

Lifelogging on the Internet

Recently, the keeping of web based lifelogs has become common. Web based lifelogs (also called weblogs or blogs) are used to keep manually annotated text and image based lifelogs. Extensions to web logs which enable individuals to make video log entries to their weblogs are also in use and are known as vlogs. Widespread availability and use of camera equipped cellular phones and PDAs means that it is now easier for individuals to keep a record of their lives using video and audio rather than only describing their experiences textually. The term moblog has been coined to describe weblogs which are

maintained using memories captured by mobile PIM devices. Updating weblogs is as simple as sending an email message which can contain audio, video or text. A large number of freely hosted weblogs are available which enable individuals to maintain daily web log entries. Examples of weblogs include Blogger [Blogger, 2005], a free weblog that enables the making of diary entries using email messages. A number of free video/audio and general data storage services are available on the web, which means they can be used in conjunction with a weblog service such as blogger to keep a daily log as audio and video. Flickr [Flickr, 2005] is a free, picture only weblog where weblog entries are made as pictures described by text (tags) to help in subsequent image retrieval. TextAmerica [TextAmerica, 2005] is a large web based community of individuals who maintain web logs using text, audio and video. Log entries can be searched using text tags. Mobilblogg [Mobilblogg, 2005] is an example of an audio/video blog which allows individuals to email their memories to their logs from their cellular phones.

2.2.3.2 Lifelog systems in research which go beyond current lifelog systems

Recently, there have been calls from institutions that urge scientists to conduct research into systems which manage lifelong memories. Fitzgibbon [Fitzgibbon *et al*, 2003] describes the UK Challenge [UK Challenge, 2005], which raises awareness of the problem of lifelong memory management. Other programs which are set up to investigate lifelogging include those from ATR [ATR, 2002] and DARPA [DARPA lifelog, 2003]. The first workshops on the capture, archival and retrieval of personal experiences (CARPE) [CARPE, 2004] and the 2nd international pervasive computing conference [Pervasive, 2004] were held in 2004. There is also a planned publication of a special issue on memory and the sharing of experiences [PUJ CFP, 2004] on the journal of personal and ubiquitous computing [PUJ, 2005] in 2006. All of the above initiatives show that there is a realization by researchers of the need to develop methods and tools to help manage the memories formed throughout a person's life.

Image based lifelogging

Mann [Mann, 1997, 2001] claims to be one of the first people to start lifelogging through the capturing and transmission of images wirelessly to a remote location. He made

available images captured by his wearable camera onto his web server which meant that people who are connected to the Internet can also see what he is seeing. In one instance, he reports a fire breaking out on his campus and how his photographic transmission made it possible for other people to also follow the event remotely. Besides Mann, a number of individuals have constructed experimental platforms and devised experiments in order to investigate lifelogging.

Audio based lifelogging

Ellis [Ellis *et al*, 2004], Kern [Kern *et al*, 2004], [Tsegaye *et al*, 2004] and Vemuri [Vemuri *et al*, 2004] have developed audio only lifelogs. Audio is easy to capture because sensors which record audio (microphones) are widely available in cheap, portable and power efficient form. All of the authors except Kern use a microphone in combination with a PDA to capture audio. Kern processes captured audio to identify speakers, Vermuri and Ellis extract text from captured audio using text to speech converters in order to use this information to facilitate the retrieval of audio. All the authors use text annotation to facilitate audio retrieval.

Video and audio based lifelogging

Hoisko [Hoisko, 2003], Deutscher [Deutscher *et al*, 2004] and Frigo [Frigo, 2004] develop video/audio lifelogs. All authors use digital cameras and a microphone for capturing memories and text annotations to retrieve captured memories. Frigo created a portable image based lifelog that used a PDA for storage and retrieval of memories, but restricts captured memories to objects he interacts with. Hoisko placed his digital cameras on his body to capture video/audio memories and calls his setup a wearable computer. Deutscher set up cameras in social gatherings to record a person's experiences in order to evaluate their reactions to his recordings and to allow them to view captured memories during the social gathering at the time of capture.

Lifelogging that includes memories other than video and audio

Aizawa [Aizawa, 2001, 2003, 2004], Clarkson [Clarkson, 2002] and Gemmell [Gemmell *et al*, 2005] develop lifelogs that capture memories besides audio and video using additional sensors. Aizawa's and Gemmell's aim of capturing data with sensors is to

analyze and use information from sensors to tag and help retrieve captured memories, whereas Clarkson is interested in using sensor data to create an automatic activity and location recognizer which learns from past sensor data to predict future user activity and location rather than to keep a lifelog. His memory capture setup shows how a limited number of sensors can be packaged in a wearable form to be used to capture some personal memories. Gemmell also develops a wearable memory capture device (necklace) which contains a number of sensors. Gemmell and his group [Gemmell *et al*, 2005] are in the process of developing Mylifebits, a lifelong memory management software for a desktop computer which helps people keep track of all of their memories as well as enable them to share them with other people.

Amongst all the works considered above, there is a realization that it is difficult to study machine-based personal memory management over long periods of time. Firstly, there are hardware limitations imposed by current technology which make it difficult to capture all of a person's memories. Secondly, no software has yet been developed to enable the efficient capture, storage and retrieval of lifelong memories. Moreover, in order to evaluate the effectiveness of such a system the evaluation would have to run over long periods of time, such as a the lifetime of a person.

2.2.4 Platforms for future machine-based memory managers

It is difficult to predict what will happen in the future, but by considering trends in past and current use of technology, it may be possible to identify certain technologies which are likely to be adopted by individuals in the near future for management of their memory. Kjeldskov [Kjeldskov, 2003] sees these platforms as being a move away from traditional platforms such as desktop computers and calls them emerging technologies for human computer interaction.

2.2.4.1 Past and present platforms for machine-based memory managers

In the past a variety of electronic devices have been used for personal memory enhancement and management as shown by Koblentz's [Koblentz, 2005] and Rhodes's [Rhodes, 1997] accounts. Both authors construct timelines of PIM devices used by

individuals in the past. Koblentz's timeline shows handheld, self-powered, personal digital assistant devices in use from the 1970s to the 1990s. Rhodes's timeline starts much earlier, spanning from 1268, which he lists as the earliest recorded date of eyeglass use, to the late 1990s when wearable wireless webcams were used to transmit images from a point of capture to a remote location by a private individual (Steve Mann in 1994). This is in contrast to transmissions made by institutionally owned television broadcasters. In addition, currently a number of platforms for machine-based memory managers exist. These include:

- small mobile devices such as cellular phones which are currently popular. The GSM Association [GSMA, 2005] reports that close to two billion are currently in use.

- wearable computers, various implementations exist [McCarty, 1999]. Examples are shown in Figure 3. The concept of wearable computers is that computing devices will shrink in size and be cheap to manufacture in the future thus they can be incorporated into an individual's everyday clothing [Mann, 1998].

- thin clients such as small hand held displays (tabs) described by Weiser [Weiser, 1991] and the personal data server described by Want [Want *et al*, 2002] which are connected by a network such as the Internet.

2.2.4.2 Platforms for machine-based memory managers in the distant future

The ultimate machine-based personal memory management platforms are those which interface directly to the human nervous system and draw their energy from the human respiration system. Such machines become extensions of an individual's body and mind. This kind of extension of the human body with artificial devices has been described by authors such as Mann [Mann, 2001] and Warwick [Warwick, 2005] as the turning of humans into cybernetic organisms, organisms which are part machine part human. To some extent, humans already extend their abilities with artificial devices such as eyeglasses, hearing aids, heart pacemakers and artificial limbs. In the future, cybernetic devices will enable humans to attain superhuman attributes such as photographic memory

through artificial organs which manage and enhance the human memory system. Their operation would be transparent to the individual as such organs are part of the body and use the body's biological mechanisms to maintain themselves.

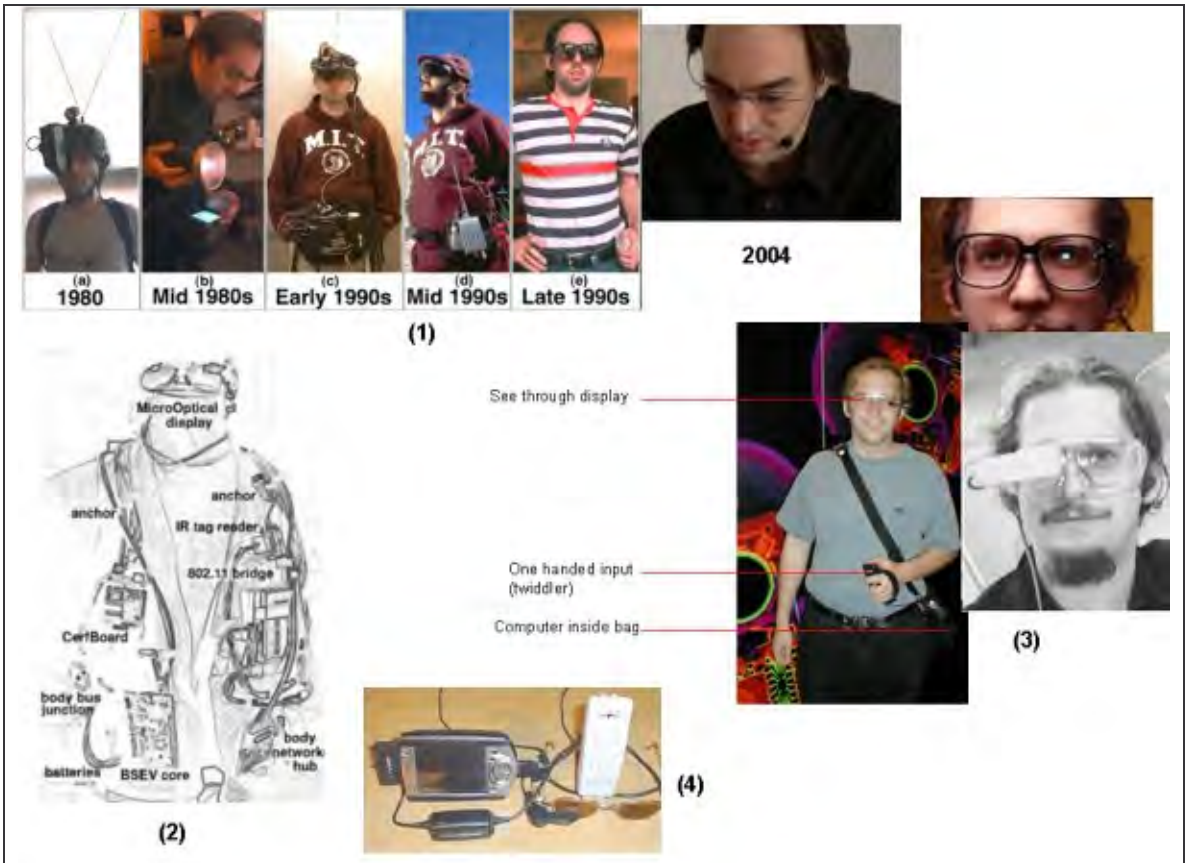


Figure 3. Various Wearable Computers (1) Evolution of Mann's wearables (2) MIT's wearable [MIThril, 2003] (3) Starner's wearables [Starner 1999] (4) Ackermann's [Ackermann, 2001] PDA based wearable

Another future platform for machine-based memory managers which is important to consider are robots. These are fully artificial machines which observe and assist individuals in the same way in which current human assistants do but with improved efficiency. Hanson [Hanson *et al*, 2005] has recently demonstrated the potential of android-like machines by building a realistic robot which makes available some of the memories of the science fiction writer Philip K. Dick to people who have not met him. Robots can be made into entities which are fully autonomous and mobile. They can offer services, such as management of a personal memories, act as companions and provide personal security. Robots can be an alternative to turning humans into cybernetic

organisms but are not transparent since they are entities which are separate from the individual.

2.3 A comparison of existing machine-based memory aid research

In this section, a comparison of the works considered in section 2.2 is performed. Ideal characteristics that a machine-based memory manager should have are identified and used as criteria for comparing the various works. Additional criteria for the comparison are also established. The comparison is then performed and areas in which this work can make contributions in are identified.

2.3.1 Ideal characteristics of a machine-based personal memory manager

The various works considered in section 2.2 identify a number of characteristics that machines such as a personal memory manager should have. Authors propose that machine-based memory managers should

- be transparent and attentive [Weiser, 1991] [Mann, 2001]
- act as a memory appliance [Norman, 1998]
- act as a memory prosthesis [Lamming *et al*, 1994]
- augment an individual's senses [Mann, 2004] [Kawamura *et al*, 2002, 2004]
- be able to protect the privacy of the individual by ensuring that managed memories are only accessible by their owner or under the control of their owner [Mann, 2001], [Vemuri *et al*, 2004]
- be an intimate entity that accompanies a person everywhere [Lamming *et al*, 1994], [DeVaul *et al*, 2000], [Mann, 2001]
- be self-sustaining in term of energy use [Starner, 1999], [Mann, 2001]

The comparison conducted in this section can show whether authors integrate one or more of the above characteristics in their machine-based memory aids.

2.3.2 Additional criteria for the comparison

In addition to considering the ideal characteristics of a machine-based memory manager, a set of criteria that can be used to compare the works considered in section 2.2 can be established by identifying important aspects of machine-based personal memory management approaches examined thus far. These include:

2.3.2.1 Kinds of personal memory management investigated

Researchers whose works have been examined focus on various aspects of personal memory management including short-term memory, long-term memory, memory augmentation, user interface related issues and one or more of the processes which are part of memory management such as memory capture, memory encoding and memory storage. Some authors identify problems and present concepts but do not provide experimental work. The comparison can show the kinds of personal memory management performed by the various authors.

2.3.2.2 Memory management model considered and/or developed

When attempting to develop a machine-based memory manager to support and augment the human memory system, an understanding of the way the human memory system works is important. This can be achieved by examining previously proposed models that theorize how the human memory system functions. In addition, conceptualisation and development of a model that explains how personal memory management can be facilitated by a machine is necessary. The comparison can show if the authors develop and/or consider existing models.

2.3.2.3 Characteristics of experimental platforms used

Ideal characteristics of a machine-based memory manager are identified in section 2.3.1. Experimental platforms used by the various authors can be compared based on which of these characteristics they possess. In addition, the hardware and software platforms used can be considered. The hardware platform can be considered in terms of the platforms for machine-based memory aids identified in section 2.2.4 and the software platform can be considered in terms of the functionality it attempts to provide.

2.3.3 A comparison of the surveyed works

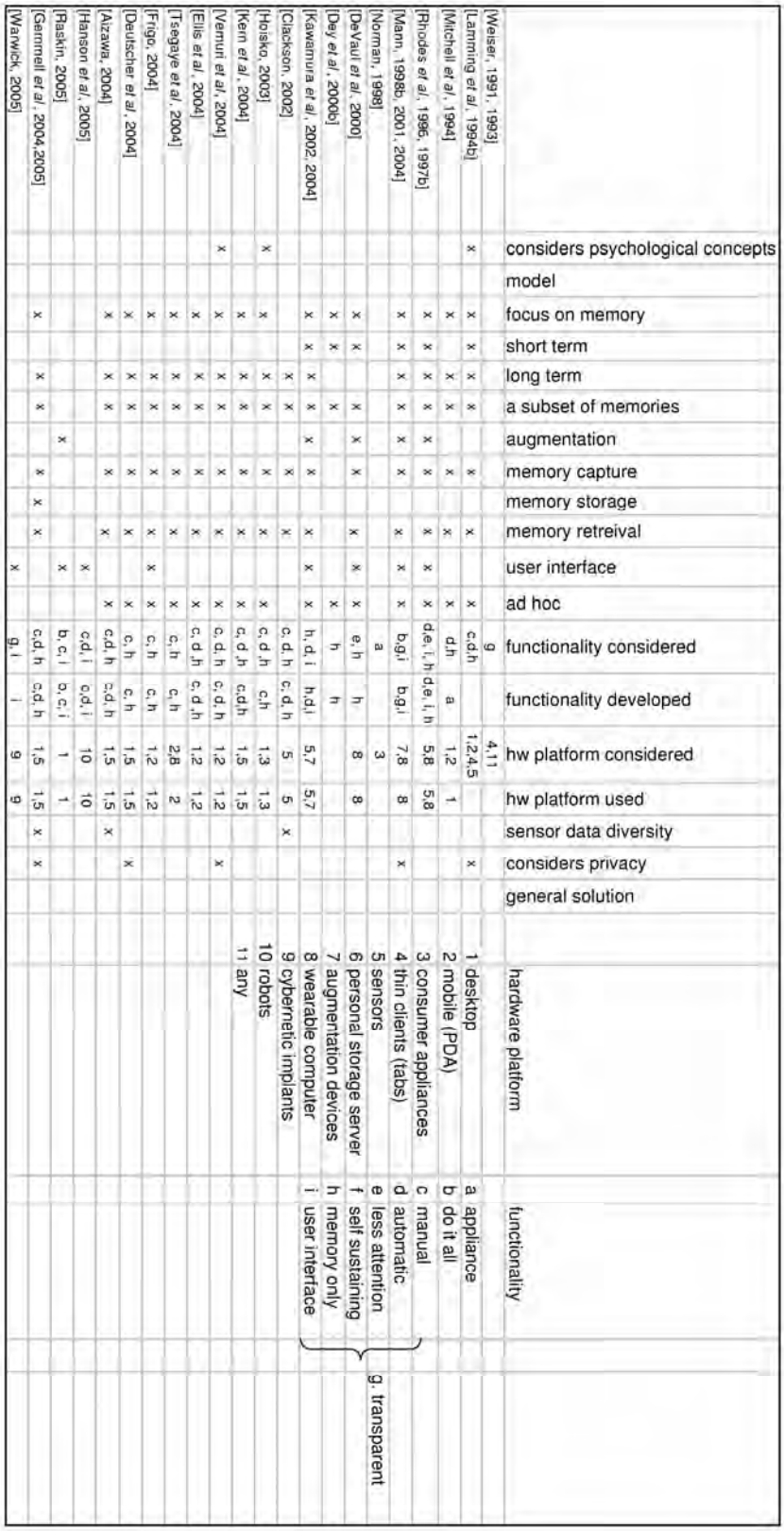
The comparison of the works which are considered in section 2.2 is shown in Table 1. The figure presents each work vs. the criteria which are identified in section 2.3.1 and 2.3.2. Works from Hanson, Warwick and Weiser are included in the comparison although they do not explicitly focus on memory aids because they provide important examples of user interfaces and hardware platforms for machine-based memory management such as thin clients, cybernetic implants and robots. Norman's and Raskin's works are included in the comparison because, like Weiser, they develop concepts that represent a new paradigm for human computer interaction such as machines which operate transparently and primarily act as appliances that provide useful facilities to people.

2.3.3.1 Characteristics of a machine-based memory manager

The comparisons shows that no single author has produced an implementation of a machine-based memory manager that integrates all of the characteristics of a machine-based memory manager which are identified in section 2.3.1: a personal memory manager which is intimate, transparent, attentive, acts as a prosthesis, augments an individual's senses, protects individual privacy and is self sufficient in terms of power use.

2.3.3.2 Kinds of personal memory management investigated

Most of the works compared in Table 1 focus on the management of a subset of long-term memories (audio/video) rather than any type of memory. There is no clear separation of the processes involved in managing memory (capture, encoding and retrieval). There is more focus on capture and retrieval of memories and less focus on investigating how managed memories are encoded and stored to facilitate their management over long periods of time.



	considers psychological concepts	model	focus on memory	short term	long term	a subset of memories	augmentation	memory capture	memory storage	memory retreival	user interface	ad hoc	functionality considered	functionality developed	hw platform considered	hw platform used	sensor data diversity	considers privacy	general solution	hardware platform	functionality
[Weiser, 1991, 1993]													g		4,11						
[Lamming *et al.*, 1994b]	x		x	x	x	x		x		x		x	c,d,h		1,2,4,5					1 desktop	a appliance
[Mitchell *et al.*, 1994]			x		x	x		x		x		x	d,h	a	1,2	1				2 mobile (PDA)	b do it all
[Rhodes *et al.*, 1996, 1997b]			x	x	x	x		x		x		x	d,e, i, h	d,e, i, h	5,8	5,8				3 consumer appliances	c manual
[Mann, 1998b, 2001, 2004]			x	x	x	x		x		x		x	b,g,i	b,g,i	7,8	8		x		4 thin clients (tabs)	d automatic
[Norman, 1998]													a		3					5 sensors	e less attention
[DeVaul *et al.*, 2000]			x	x		x	x	x		x	x	x	e, h	h	8	8				6 personal storage server	f self sustaining
[Dey *et al.*, 2000b]			x	x		x						x	h	h						7 augmentation devices	h memory only
[Kawamura *et al.*, 2002, 2004]			x	x	x	x	x	x		x	x	x	h, d, i	h,d,i	5,7	5,7				8 wearable computer	i user interface
[Clarkson, 2002]				x	x	x		x		x			c, d, h	c, d, h	5	5	x			9 cybernetic implants	g. transparent
[Hoisko, 2003]	x		x	x	x	x		x		x		x	c, d, h	c,h	1,3	1,3				10 robots	
[Kern *et al.*, 2004]			x	x	x	x		x		x		x	c, d ,h	c,d,h	1,5	1,5				11 any	
[Vemuri *et al.*, 2004]		x	x	x	x	x		x		x		x	c, d, h	c, d, h	1,2	1,2		x			
[Ellis *et al.*, 2004]			x	x	x	x		x		x		x	c, d ,h	c, d ,h	1,2	1,2					
[Tsetsaye *et al.*, 2004]			x	x	x	x		x		x		x	c, h	c, h	2,8	2					
[Frigo, 2004]			x	x	x	x		x		x		x	c, h	c, h	1,2	1,2					
[Deutscher *et al.*, 2004]			x	x	x			x		x		x	c, h	c, h	1,5	1,5		x			
[Aizawa, 2004]			x		x	x		x		x		x	c,d, h	c,d, h	1,5	1,5	x				
[Hanson *et al.*, 2005]											x		c,d, i	c,d, i	10	10					
[Raskin, 2005]							x				x		b, c, i	b, c, i	1	1					
[Gemmell *et al.*, 2004,2005]			x	x	x	x		x		x		x	c,d, h	c,d, h	1,5	1,5	x	x			
[Warwick, 2005]			x								x		g, i	i	9	9					

Table 1. A comparison of previous work in machine-based memory aids

There is a consensus amongst the compared works on the need to use information which is part of a person's environment to index their memories. This is in contrast to current personal memory management software, such as desktop search engines which index documents by video/audio properties (width and height), the words they contain and in some cases their meaning as well as human supplied document properties through text annotation (metadata). Such systems do not capture information that describes a person's states and the environment inside or outside of a desktop computer which describes the circumstances of formation of documents and use it to index documents. Although the use of environmental information for indexing a person's memories is advocated, there is less focus on using a variety of sensors and a diverse set of sensor data classifiers as sources for environmental data which can be used to tag and facilitate the retrieval of personal memories. Text annotation and location information are often used to facilitate this process.

2.3.3.3 Memory management models considered and/or developed

A few of the authors consider models of human memory, such as psychological models, and in each case their treatment is minimal. None of the authors attempt to develop a model that can be used to build a machine-based memory manager. Thus a common approach is not taken in constructing experimental systems.

2.3.3.4 Characteristics of experimental platforms used

Each type of hardware platform advocated for or used in experiments by the authors and the functionality provided by the software platform are listed in Table 1. The comparison shows that the surveyed works make use of a variety of hardware platforms which provide various levels of functionality. However, a majority of the experimental platforms are constructed on an ad-hoc basis which can be attributed to not having a common model for building a machine-based memory manager.

2.3.4 What is lacking in existing approaches

The comparison of existing research performed in the previous section shows that a general solution for machine-based personal management is lacking. This work sees such a solution as that which:

- considers existing models of memory management, such as psychological models of human memory
- considers management of all of a person's memories and not just a subset of memories
- develops a model that identifies entities and considers processes involved in machine-based memory management and explains how a machine-based memory manager can be constructed
- makes use of an experimental platform that supports experimentation in a non-ad-hoc manner (develops one if necessary). An example of non-ad-hoc experimentation is one that uses a machine-based memory manager which has the characteristics identified in section 2.3.1 and contains a diverse set of sensors and sensor data classification components. After an experiment, the experimental platform is not discarded but is continually improved

The works surveyed in this chapter do not follow the above steps thus they are not positioned well to consider the larger problem of machine-based memory management.

2.4 Towards the development of a machine-based personal memory manager

Today's appliances and computing systems such as "do it all" general purpose computers are not entirely bad. What is lacking is an entity which is transparent and is part of an individual's life. The entity would need to capture and facilitate the retrieval of memories so that the individual can make use of devices such as desktop computers and consumer appliances to manipulate them. In addition, this entity could act as a memory appliance which augments the individual's biological memory by allowing him to capture and

process memories in ways that his biological system is not capable of. This entity is referred to in this chapter and the rest of this thesis as a machine-based memory manager. However, before such an entity can be realised a number of problems remain to be addressed.

2.4.1 Finding an effective means of managing an individual's external memories

This work sees a machine-based memory manager as an entity that consists of multiple artificial sensors and components that analyze sensor data in order to extract and reason on information that can be used to manage an individual's memories. The kind of reasoning referred to here is not that studied by artificial intelligence communities in their attempts to endow machines with human-like intelligence, but rather reasoning on information which describes the states of people and their environment obtained by analysing data from sensors. Such reasoning can, for example, be used to locate personal memories provided that the memories are tagged with information that describes the circumstances of their formation. Context aware computing provides concepts which can be used to manage a person's memories in this manner and is considered in chapter 3.

2.4.2 Developing a model for a machine-based personal memory manager

In order to develop a model that can be used to construct a machine-based memory manager, existing models that can provide the abstractions necessary for explaining how a machine-based memory manager can perform its functions, need to be considered. For this reason, psychological models of human memory were considered in section 2.1 and context aware computing will be considered in chapter 3. The model should identify entities and processes which are involved in machine-based human memory management. The processes and the roles of the entities which participate in these processes should be explained. Such a model will be developed in chapter 4.

2.4.3 Testing the model

After developing a model for machine-based memory management, it is necessary to test the model. One way of testing the model is to develop an experimental machine-based memory manager which is based on the model and evaluate it. Existing hardware platforms and software tools that can be used to construct a machine-based memory manager need to be identified and/or developed. However, the experimental system should not be constructed on an ad-hoc basis. It is important to develop a method of building experimental systems which can simplify the complexity involved in building systems that have sensing and reasoning abilities. Once developed, the method can be used to aid machine-based memory management research in the future. Chapter 5 conceptualises and develops a software infrastructure which provides a method for building complex systems. Chapter 6 presents the implementation of an experimental machine-based memory manager using the newly developed software infrastructure and chapter 7 presents its evaluation.

3 Integrating context awareness into the machine-based memory manager

Context aware computing is an area of research in computer science which aims to enable computers to make use of information that describes the states of people and their environment, to offer services to people. This chapter considers a number of aspects of context aware computing. During the research period of this work, context based information management has been found to be an effective solution for machine-based personal memory management, specifically in the management of external memories that accumulate over long periods of time. The existing works which are considered in this chapter support this view.

In this chapter, the nature of context and context awareness in applications as well as current approaches at modelling context are examined in order to establish an understanding of context and context awareness. Previous proposals for memory aids which make use of context based memory management are examined. As this work aims to develop an experimental context aware machine-based memory manager, a number of proposed architectures for context aware applications are considered and features which are common to all context aware applications are identified. The context aware application development process is examined. Problems which are inherent in context aware application development are identified. A decision is made on how best to adapt existing context aware architectures for use in developing a machine-based memory manager. The need for a software infrastructure which supports the development of a context aware machine-based memory manager is identified.

3.1 The nature of context

Context has different interpretations depending on what discipline it is used in. For example, in archaeology it means the location of a discovery and in communications and linguistics it is defined as the discourse which surrounds a language unit and helps to determine its interpretation. In computer science, context also has a different

interpretation. After consideration of previous proposals for definitions of context made by various authors which he states are too specific, Dey [Dey, 2000] offers a general definition of context as *"any information which can be used to characterize the situation of entities which are considered to be relevant to the interaction between a user and an application, an entity being any person, place or object"*. Winograd [Winograd, 2001] considers this definition as too broad and sees context as information which helps in the interpretation of an action performed by a user or device and not just any relevant information, since what is important is establishing an efficient means of communication between individuals and the devices they use. Winograd states that there is a lack of agreement on what context should include. He asserts that context includes the outside world which the user is in and a system under use, not just the system under use as it is in current systems. Schmidt [Schmidt, 2002] defines context as a description of the current situation on an abstract level. Such a description can be matched to a previously specified situation.

3.1.1 A definition of context in machine-based personal memory management

The dictionary definition of context is, "The set of facts or circumstances that surround a situation or event" [WordNet, 2005]. Context is not a single item, it is a non-empty set. It represents a frame of interpretation. In this thesis, context is defined as the set of circumstances that describe a collection of external memories. More specifically, *context is defined as the set of states of a person and his environment which describe a collection of external memories.*

New terminology for context information

This work proposes the term *context descriptor state* to refer to the members of the set of states of a person and their environment (i.e. context) and it proposes the term *context descriptor* to refer to a class of context descriptor states. This definition works well as the dictionary definition of a descriptor is "a piece of stored information which is used to identify an item in an information storage and retrieval system" [WordNet, 2005]. The term *context descriptor* is non-existent in the context aware computing literature

surveyed during this work. The term context information is more prevalent. However, the use of the term context descriptor is favoured in this work because the term descriptor is more relevant to this work than "information" which could describe anything. In descriptions of the various authors' work presented in this chapter the term context information has been substituted with the term context descriptor to clarify the discussions.

3.1.2 Categorization of context descriptors

In order to identify what constitutes the states of a person and his environment it is useful to consider the existing categorizations of context descriptors which are shown in Table 2. The categorizations show the agreement by a number of authors that (1) context is made up of a collection of context descriptors and (2) context descriptors are information which describes the states of people and their environment. However, they do not make the distinction between context descriptors and context descriptor states as has been made by this work. In addition, none of the categorizations classify all types of context descriptors as it is difficult to decide what represents an adequate classification of the large amount of information which exists in a person's environment

3.1.3 Context awareness

Context awareness refers to an ability which applications have for determining context by considering a number of context descriptors. Dey [Dey, 2000] states that if an application makes use of a certain type of context descriptor, by monitoring changes in the descriptor's states, to offer services to users, it is said to be context aware. Awareness of time and identity has, for example, been used in computers for a long time. Calendar applications are based on the awareness of time. Logon and computer resource access in an operating system is controlled by an awareness of user identity. However, use of these two context descriptors, on their own, is not enough.

Dey [Dey, 2000] identifies three facilities which context aware applications can provide: (1) presentation of information and services to a user, (2) automatic execution of a service and (3) tagging information with context descriptors for later retrieval. In this

thesis, the third feature is seen as being the most important facility which applications can support.

Author	Categorization of context descriptors
[Schilit *et al*, 1994], [Brown, 1998]	the states of the immediate environment that are picked up by a person's senses which help people in their interactions with devices • computing resources (e.g. processors, input devices, network), • the user environment (e.g. location, people, social situation) and physical environment (e.g. light level, noise)
[Korkea-aho, 2000]	almost any information available at the time of interaction with services that helps describe the context of interaction • identity • spatial information - e.g. location, orientation, speed and acceleration • temporal information - e.g. time of the day, date and season of the year • environmental information - e.g. temperature, air quality and light or noise level • social situation - e.g. who a person is with and people who are nearby • resources which are nearby - e.g. accessible devices and hosts • availability of resources - e.g. battery, display, network and bandwidth • physiological measurements - e.g. blood pressure, heart rate, respiration rate, muscle activity and tone of voice • activity - e.g. talking, reading, walking, and running
[Schmidt, 2002]	• information related to human factors including user's habits, psychophysical states, interaction with people, group dynamics, activities, tasks and general goals of people. • information related to the environment including user location, environmental conditions and infrastructure.
[Aizawa, 2003]	• objective context descriptors such as time, location and behaviour of an individual • subjective context descriptors such as feelings and interests of a person
[Haya *et al*, 2004], [Dey, 2000]	• primary context descriptors which consist of three entities: a place (has environmental properties), a person (has activity and identity) and a resource. • secondary context descriptors which consist of domain specific context descriptors that extend the primary context types by adding more entities relevant to a model.
[Bristow *et al,* 2004]	• five groups: event, environment, task, person and object. These context descriptors, he says are useful determinants of user context and are validated by his user evaluation

Table 2. Various categorizations of context descriptors

If applications have the ability to use context descriptors to tag information then they can support the first and second features. If they can easily retrieve information using context descriptors, then they can present the right information and services to a user or execute a service based on a user's context.

Existing context aware applications

Korkea-aho [Korkea-aho, 2000] and Mitchell [Mitchell, 2002] present surveys of various types of context aware applications which have been developed in the past. These include tour guides, memory aids, office assistants, meeting tools and smart environments. Both authors conclude that (1) existing context aware applications are prototypes developed in research labs and the academic world and (2) there is complexity in implementing such applications because they make use of multiple sensors and components that analyze sensor data in order to identify context descriptors. Mitchell further asserts that there is a lack of generic mechanisms for supporting mobile and context aware applications.

3.1.4 Context modelling

Context modelling involves forming a relationship between sensor data and real world knowledge. Parkin [Parkin, 1999], for example, states that for cognition the human memory system uses a model of the real world which is based on previous experiences of an individual. Context aware applications, on the other hand, need a context model for inference. Thus, when developing context aware applications some amount of context modelling is necessary.

In a survey of context modelling approaches taken by a number of authors, Strang [Strang *et al*, 2004] identifies a number of approaches: (1) key-value such as Schilits's approach [Schilit, 1995], (2) mark-up schemes which use hierarchical data structures such as Heckmann's [Heckmann, 2003] approach, (3) graphical models which use modelling languages such as UML, (4) object oriented models such as Schmidt's cue based abstraction [Schmidt, 2002], (5) logic based models where context is expressed as facts, expressions and rules, and (6) ontology based modelling. Ontology based modelling is currently popular. Christopoulou [Christopoulou *et al*, 2004], Millard [Millard *et al*, 2004], Gu [Gu *et al*, 2005] and Schraefel [Schraefel *et al*, 2005] all use semantic web technology from the W3C [W3C, 2001] for modelling context. Strang states that ontology based modelling is the most promising approach to use when modelling context. He points out that instead of developing context models which are

aimed at single applications, developing generic context models is important because the model can benefit many applications.

In this work, a context modelling approach which uses collections of key-value pairs, where the key is a context descriptor and the value is a context descriptor state, as groups called context profiles, is used. It is introduced in chapter 4.

3.2 Existing context aware memory aids

Context aware memory aids are applications which make use of context descriptors in order to offer facilities that aid a person's biological memory system. In this section, systems that use text supplied by users to describe user context are presented. The practice of using text supplied by users (text annotation) which contain context descriptors to index a person's memories is still prevalent in today's systems.

3.2.1 Rhodes' remembrance agent

Rhodes [Rhodes *et al*, 1996] proposes a system for augmented memory called the remembrance agent, which displays a list of documents that it identifies are of interest to a user. It does so by monitoring a user while he is using a text editing application. The words that a user types are matched with the text which is contained in a document. If a match occurs, the document is displayed as part of a list of documents which contains the same words.

Rhodes later extends his system to a wearable form [Rhodes, 1997b] calling it the wearable remembrance agent. He proposes that sensors should be part of the wearable computer and can be used to detect personal situations. This new wearable system uses context descriptors to suggest documents which it identifies as relevant to the current user context. It uses a number of context descriptors: wearer's physical location, people that are currently around, subject field and time stamp to suggest text documents. When users enter new notes, location and time stamp context descriptors are used to annotate them automatically if available from hardware sensors. If hardware is not available, users can provide context descriptors through text annotation. Large text documents are broken

down into smaller documents, which make it easier to search for matches by words. The general approach of finding documents by text based searching which is used by Rhodes is currently widely used by search engines such as Google. Rhodes' wearable remembrance agent runs on Starner's Lizzy computer [Starner, 1999] and it is entirely a text based system, including its user interface which consists of the Emacs text editor.

Rhodes [Rhodes, 2003] performed an evaluation to determine the usefulness of a user's physical context for automatically retrieving archived notes. The archive contains more than 950 notes (640 are associated with context descriptors) spanning from 1996 to 2000. From the results of his evaluation he observes that physical context is not especially useful for automatically retrieving information from an archive but asserts that since the notes came from one person only, it cannot be assumed that context descriptors are not useful for helping in retrieving information.

3.2.2 DeVaul's memory glasses

DeVaul [DeVaul *et al,* 2000] presents work on a prototype wearable reminder, called the memory glasses, which he intends to be situation appropriate, a reminder that works with no input except an initial request to be reminded. He aims to design a reminder to act as a short term memory aid which requires minimum user attention comparing it to a system which would replace a reliable human secretary. DeVaul categorizes reminders which use context descriptors into time, location/time, activity/location/time based reminders. He asserts that location is important because it offers clues to user activity which he claims is independent of time and allows systems to adjust their behaviour depending on changes in activity. He sets as his goal the construction of an activity, time and location based reminder which can recognize user activities and environmental conditions through training. DeVaul recognizes that there is a difficulty in recognizing user actions and environmental conditions which require high bandwidth sensing and sophisticated analysis.

DeVaul's wearable system consisted of a camera (USB camera), microphone, a laptop computer (with a 266 MHz CPU) and user interaction peripherals (track pad, headphones

for audio output). A user interacts with the wearable system using multiple single stroke gestures. Audio is played to provide user interaction feedback. A see-through wearable display is used to display information to a user. The software running on the wearable computer consists of a dual booting operating system (a mix of Windows and Linux), device drivers and the components listed below.

- A classifier system, which is a signal processing system that converts raw sensor data into a collection of probabilistic estimates. It allows a user to train event recognition functions to allow patterns to be recognized and tag them as specific events. A Hidden Markov Model (HMM) grammar capable of recognizing patterns from a range of time scales is used.
- A perceptual context classifier which extracts basic features from audio and video (spatial moments in video, audio volume and amount of speech).
- Memory agent layer: an agent oriented layer, sits on top of the classifier system. It contains an agent orientated programming layer and memory glasses application layer. The middleware is based on the HIVE agent system created by Minar [Minar *et al*, 1999]. HIVE consists of a collection of agents working together. The HIVE programming environment allows data flow views of the memory glasses application. DeVaul states that this allows for interaction flexibility.

DeVaul concludes that his system can be useful as a proactive context aware reminder system and set as a goal the production of an ergonomic and useful memory aid tool. DeVaul, however, has since changed his focus [DeVaul *et al*, 2003] to a different kind of human memory improvement called subliminal queuing where the aim is to improve a person's biological memory by flashing images that contain messages using a specialized experimental setup.

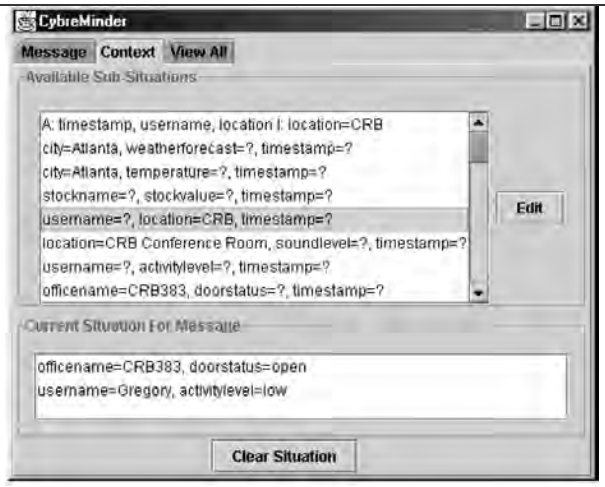
3.2.3 Dey's CybreMinder

Dey [Dey *et al*, 2000b] states that current tools do not provide enough support for reminders. Dey lists paper to-do lists, emails, post-it notes, electronic PIMs and human assistants as examples of reminder tools, none of which he claims make use of rich sets

of context descriptors. As a solution, Dey proposes CybreMinder, which is a context aware system for supporting reminders. He provides the following as a list of features that ideal reminders should have:

- use of rich set of contexts
- submission of reminders by third parties
- submission of reminders using a variety of input devices
- situation appropriate notification via a variety of devices
- signal based (sound) and content based (description of event) reminding
- showing of a list of active reminders

In CybreMinder, users create reminders using a GUI application, in which they specify context descriptors as well as textual descriptions of an event (name, subject and reminder message). The CyberMinder GUI and examples of the reminders it uses are shown in Figure 4.



Situation	Natural Language Description	CybreMinder Description
Time	9:45 am	Expiration field: 9:45 am
Location	Forecast is for rain and Bob is leaving home	City = Atlanta, WeatherForecast = rain Username = Bob, Location = Bob's front door
Co-Location	Sally and colleague are co-located	Username = Sally, Location = *1 Username = Bob, Location = *1
Complex #1	Stock price of X is over $50, Bob is alone and has free time	StockName = X, StockPrice > 50 Username = Bob, Location = *1 Location = *1, OccupantSize = 1 Username = Bob, FreeTime > 30
Complex #2	Sally is in her office has some free time, and her friend is not busy	Username = Sally, Location = Sally's office Username = Sally, FreeTime = 60 Username = Tom, ActivityLevel = low

Figure 4. Dey's CybreMinder [Dey *et al*, 2000b]

CybreMinder uses context descriptors for location, identity, time and activity. A decision to remind a user is made by monitoring key-value pairs of context descriptors. A user is left to select/edit a combination of context descriptors from a list box. CybreMinder can deliver reminders based on location (when a user is at a location, he is reminded of some condition), co-location (two or more users are reminded of a condition when they are in close proximity, for example, during a meeting) as well as when other context descriptor values are matched.

3.2.4 Aizawa's context based video management

Aizawa [Aizawa, 2004] states that video recordings can be used as lifelong memory aids. He proposes that context descriptors can be used to index video recordings which are a much more efficient means of retrieving segments of video that interest users than using current content based indexing and retrieval methods. Aizawa's video management system uses a number of context descriptors to index memories, including location, orientation, motion and user interest which are shown in Figure 5. He uses context descriptor values as keys for memory retrieval.

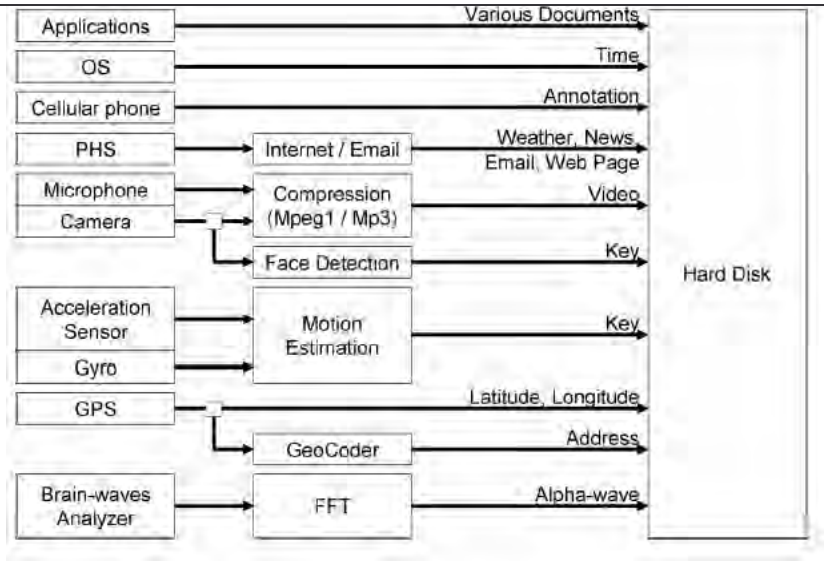


Figure 5. Aizawa's proposal for a context based memory aid [Aizawa, 2004]

3.2.5 This work in relation to existing context aware memory aids

Rhodes, DeVaul, Dey and Aizawa provide important concepts that this work can build on such as managing a person's external memories using context descriptors. They also show the relative complexity of the experimental system which is needed to investigate a machine-based memory manager which uses context descriptors to manage a person's memories. Rhodes and DeVaul focus on managing a subset of a person's memories (text or video), Dey focuses on providing PIM services that use context descriptors for information retrieval. Aizawa focuses on using context descriptors from multiple sensors to index video and electronic documents. These works, however, do not conceptualise a generic memory manager that uses context descriptors to manage external memory. They do not explain how memory fragments which accumulate over long periods of time will be managed. They do not provide a model which can be used to build a machine-based memory manager. They also do not address how the complexity of building a machine-based memory manager can be overcome. This work makes contributions in these areas.

3.3 Architectures for context aware applications

A number of context aware applications have been developed by various research groups and individuals in the past. During development, each group designs a particular architecture which is suited to their context aware application. This section compares existing architectures and identifies features which are common to all of them. It also identifies the different types of architectural components that need to be implemented in order to create context aware applications. The comparison uses Dey's architecture [Dey, 2000] as a base for comparison because he proposed one of the first architectures for context aware applications. The material which is presented in this section is important to consider because this work plans to implement a context aware machine-based memory manager which will contain some of these components.

3.3.1 Features common to all context aware architectures

In all the context aware architectures which are considered in this chapter, three basic layers are prevalent and these are: a physical layer, a context layer and an application

layer. The physical layer is concerned with the acquisition of data from a user's environment which serves as a source of context descriptors. It consists of both hardware and software components. The context layer, which consists of software components, is concerned with the extraction of context descriptors with the help of existing knowledge of a user's environment which has been established through the process of context modelling. Context descriptors are extracted using inference in a process known as context interpretation. The application layer contains context aware applications that make use of context descriptors which are provided by the context layer. On the boundary of the context and application layers, a context service/provider exists, which provides context descriptors to applications.

3.3.1.1 General architectures for context aware applications

Winograd [Winograd, 2001] identifies three system architectures which are used by individuals for implementing context aware applications. They are: (1) Dey's widgets architecture which is shown in Figure 6. Widgets are similar to GUI components. (2) A network service-based architecture where applications discover and connect to one or more context aware services and make use of the information they provide. (3) The blackboard architecture which is a model that focuses on data exchange. Messages are produced and consumed from a common central blackboard (middleware layer) which handles communications transparently to applications.

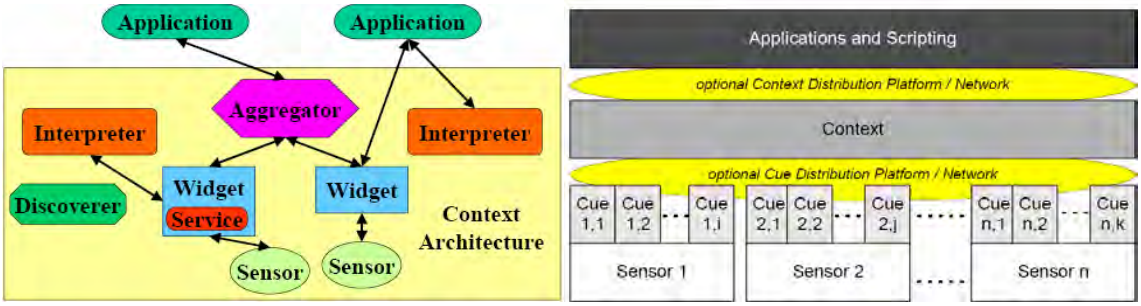


Figure 6. Dey's [Dey, 2000] and Schmidt's [Schmidt, 2002] context aware architectures

Winograd points out that the use of a networked, service-based architecture has more advantages than widgets since components used to manage context descriptors are less

dependent on each other (they are loosely coupled). The widgets model assumes that there is one application managing widgets. A fourth model, shown in Table 3, which Winograd does not consider, is Schmidt's architecture where the sensing, context and application layers are distributed throughout a network. Table 3 shows the architectures which are used by the works compared in this section.

Author	**Architecture used**
[Dey, 2000]	Widgets
[Pascoe, 1998]	Service based
[Schmidt, 2002]	Distributed fuzzy space (blackboard)
[Mitchell, 2002]	Service based
[Judd *et al*, 2003]	Service based
[Haya *et al*, 2004]	Blackboard
[Riva, 2004]	Service based
[Khedr *et al*, 2005]	Service based
[Gu *et al*, 2005]	Service based
[Bardram, 2005]	Service based
[Kawsar *et al*, 2004]	Widgets

Table 3. System architectures used by various context aware architectures

3.3.2 Physical layer

In the physical layer, Dey [Dey, 2000, 2001] provides widgets which act as sensor drivers that collect information from the environment through hardware or software based sensors. A comparison of Dey's sensor layer with sensor layers from other architectures, most of which were proposed after Dey's architecture, is shown in Table 4.

Schmidt, Haya, Khedr and Gu separate sensor drivers from components that provide context descriptors to applications such as widget, abstract context services or contextual information providers. Bardram and Kawsar introduce actuators, which are the opposite of sensors. Rather than capture sensor data from the environment, they introduce changes in environmental states.

Dey	Widgets	Capture sensor data, similar to GUI widgets
Pascoe	Sensor drivers	Capture data from the environment
Schmidt	Sensor drivers and cues (context producers)	Capture sensor data. Cues reduce sensor data e.g. by averaging data
Mitchell	Abstract context services	Capture sensor data, store persistent data
Judd	Contextual information provider	Capture sensor data, populate database
Haya	Producer	Capture sensor data, publish to blackboard
Riva	Sensor drivers	Capture Sensor data
Khedr	Agents and network probes	Capture sensor data
Gu	Sensor drivers	Physical and virtual (web services), capture sensor data
Bardram	Context monitors, Context actuators	Capture sensor data, change environmental context
Kawsar	Widgets, actuators	Capture sensor data, change environmental context

Table 4. The sensor layer in various context aware architectures as compared to Dey's physical layer

3.3.3 Context layer

In the context layer Dey [Dey, 2000, 2001] provides widgets which are comparable to GUI widgets. They have attributes and a mechanism (callbacks) for notifying applications when changes in their attributes occur. Applications can also directly access the attributes of widgets. The class of data that widgets provide, using the terminology introduced in this work, represent context descriptors and their attributes represent context descriptor states. Dey also introduces the concept of context interpreters, which he states abstract raw or low level context information into higher level forms of information. To shield applications from having to interact with many widgets, Dey introduces the concept of a context aggregator which provides a single interface to applications. A discoverer enables applications to locate and use widgets or context aggregators. Table 5 shows a comparison of Dey's context layer with the context layers from other context aware architectures.

In addition to Dey's architectural components, Mitchell's and Khedr's architecture contain components that perform session management such as seamless roaming for applications as well as application constraint management such as privacy control. In the various architectures, slightly different terms are used to refer to the same architectural components.

Dey	**Widgets**	**Context interpreters**	**Context aggregators**	**Discoverer**	**Context model**		
Pascoe	contextual information service (CIS)	synthesizers and monitors	CIS		object oriented		
Schmidt	distributed Fuzzyspace (blackboard)	context interpreters			object oriented		
Mitchell	context service provider	context translation services	bespoke context services	context service provider	object oriented	privacy and application constraints	session management
Judd	contextual Information Service (CIS)		CIS		object oriented		
Haya	blackboard	interpreter			object oriented		
Riva	context provider		context service		ontology		
Khedr	directory	context inference	communication protocol	context service discover	ontology	privacy and application constraints	roaming, Session management
GU	context providers	context interpreters	context service	service discoverer	ontology		
Bardram	context monitor	context transformers				access control	
Kawsar	widgets		virtual Widget	resource manager		user preference and privacy	

Table 5. The context layer in various context architectures compared to Dey's context layer

3.3.4 Application layer

In Dey's architecture, applications acquire context descriptors from widgets or context aggregators which applications access through a discoverer. Applications poll the attributes of widgets or subscribe for notification in context changes. Table 6 shows how applications access context descriptors in other architectures compared to Dey's architecture.

Table 6 shows that the same mechanism is used in every architecture to provide context descriptors to applications. However, applications interact with different architectural components such as widgets, a service or a blackboard.

Dey	Polling or subscription for notification through widgets
Schmidt	Through distributed communication platform (context consumers)
Pascoe	Querying the contextual information service
Mitchell	Subscription for notification from context service provider
Judd	Querying the context information service (database)
Haya	Applications consume data from blackboard
Riva	Query, polling, subscription through context providers
Khder	Query directory (cache updated by context providers) or context provider found through context discoverer and event based notification or invocation of service
GU	Through context providers or through context discoverer. Query or subscription for events
Bardram	Querying and subscription based
Kawsar	Querying based

Table 6. Context descriptor access by applications in other context architectures compared to Dey's architecture

3.3.5 Implemented applications

Mitchell implemented a tourist guide system called GUIDE that provides information to tourists based on their location. Haya implemented a smart home with three categories of applications: (1) access control with email notification, (2) personal identification (services which depend on the identity and presence of individuals in a room such as turning lights on when the first person enters a room or greeting people through text to speech) and (3) a meeting aware application which notifies people by email when a meeting takes place. Khedr implemented a prototype conference application which arranges meetings based on location. He used an RF-tag system to provide identity and location context descriptors. Gu implemented a prototype smart home which provided a number of facilities including (1) a happy dining room because the dining room plays music and adjusts lighting when a person is in the room, (2) home energy saving because lights are turned off when a person leaves a room and (3) smart phone service which diverts calls to a voice mailbox when a user is not available and increases the volume of a phone when noise level is high. Bardram developed proximity based user authentication in a hospital environment. Kawsar developed a smart assistant application using his architecture which consisted of what he calls *sentient objects* (represented by virtual widgets) which are aware of their usage and interaction history, for example, chairs which know who uses them or dishes which know what is put in them.

3.4 Developing context aware applications

Developing context aware applications refers to creating working prototypes of architectures and applications which are based on the architectures that are considered in section 3.3. Even though the various architectures discussed in the previous section shield applications from the physical layer and context layer, difficulties still remain in developing context aware applications. This difficulty needs to be addressed before developing a context aware machine-based memory manager.

3.4.1 The context aware application development process

The works considered in section 3.3 describe how the various components of a context aware architecture can be constructed. Individuals need to implement the physical, context and application layers of the system or adopt existing implementations of context aware architectures for use in their application. Even though individuals whose works have been considered have developed an API (library) which enables applications to make use of their architecture, building an experimental context aware system still requires identifying/building sensing hardware, identifying/implementing sensor drivers, context interpreters, context aggregators, building a context model, and implementing applications that understand how to use context descriptors.

3.4.2 Problems inherent in developing context aware applications

Developing context aware applications is a difficult task [Salber *et al*, 1999], [Korkea-aho, 2000], [Dey, 2000], [Mitchell, 2002]. Schmidt [Schmidt, 2002], Laerhoven [Laerhoven, 2002] as well as the discussion in Appendix F show that building experimental sensing hardware which is needed in context ware applications is time and resource intensive. Narayanaswami [Narayanaswami *et al*, 2002] shows that building a real world application is even more difficult.

Dey [Dey, 2000] cites the following problems which are faced by context aware application developers.

- Lack of a variety of sensors for determining a given context: most context aware applications tend to make use of a limited set of sensors, such as those which sense location [Korkea-aho, 2000], [Schmidt *et al*, 2001].
- Difficulty in dealing with a variety of sensors. There is a lack of support for programmers to work with specific hardware [Pascoe, 1998], [Mitchell, 2002].

The above problems also mean that:

- there is a lack of sources of a variety of context descriptors. A variety of context descriptors are important for determining the multitude of contexts which are part of a person's life

3.4.3 The difficulty in using a variety of sources of context descriptors

A comparison of a number of sensor data classification works including those outside of context aware computing research, categorized into groups which describe the context descriptors that they can be used to identify, is presented in Table 7 (For a detailed description of each author's work refer to Appendix B). A categorization which groups context descriptors into location, physiology (internal to body), body, speech, object, identity, face, environment and document is proposed to show the type of context descriptor each work focuses on. A second categorization which includes the main senses of humans: vision (video), hearing (audio), taste, smell and touch as well as thought is also proposed in order to show how much focus is put into identifying context descriptors from these senses. The comparison also identifies authors who use wireless sensors and those who use mobile platforms. The software column indicates if a sensor data classifier is available as a software toolkit.

	Location	Phyiology	Body	Speech	Motion	Objects	Identity	Face	Environment	Documents	Multi Sensor	Video	Audio	Taste	Smell	Touch	Thought	RF	Context Aware	Software	Mobile
[Kim *et al*, 2004]										x		x									
[Nelson, 2002]						x						x									
[Rekimoto *et al*, 2000]	x					x	x					x									
[Rohs *et al*, 2004]						x						x									x
[SemaCode Corp, 2005]	x					x						x									x
[Ueoka *et al*, 2004]						x						x									x
[Aoki *et al*, 1999]	x											x									x
[Clarkson *et al*, 2000]	x										x	x	x								x
[Hagan *et al*, 2002]			x									x									
[Goldberg *et al*, 1999], [Betke *et al*, 2002, 2004]								x				x									
[Sumi *et al*, 2004]								x				x									x
[Cheng *et al*, 2001]			x									x									x
[Stauffer *et al*, 2000]				x		x						x									
[Asada et al, 2003], [Healey *et al*, 1998]		x																			x
[Picard *et al*, 1997]		x									x										x
[Teller, 2004]		x									x										x
[Toko, 1998]			x											x							
[Wyszynski *et al*, 2005]			x												x						
[Vemuri *et al*, 2004]				x									x								x
[Hoisko, 2003]				x								x	x								x
[Scott *et al*, 2005]	x												x								
[CMU Sphinx, 2003], [Festival, 2003], [Microsoft Speech, 2005]				x									x							x	
[Rhodes, 1997]	x																	x			
[Mitchell, 2002]	x																	x	x		
[DeVaul *et al*, 2000]	x																	x			
[Bristow *et al*, 2004]	x											x						x	x		x
[Eagle, 2005]	x																	x			x
[Patel *et al*, 2004]	x						x						x					x	x		x
[Place Lab2, 2005]	x																	x		x	
[Small *et al*, 2000]	x																	x			x
[Stanford, 2003]						x	x											x			x
[Want *et al*, 2000]						x	x											x			
[Farringdon *et al*, 1999]			x		x																x
[Kern *et al*, 2002, 2003, 2004]			x		x									x					x		x
[Randell *et al*, 2000]			x		x																x
[Blinkx, 2005], [Copernic, 2005], [Google, 2005], [Microsoft, 2005]										x										x	
[Aizawa *et al*, 2001, 2003, 2004]	x	x		x	x		x	x		x	x	x	x					x	x		x
[Gemmell *et al*, 2004, 2005]	x				x				x	x	x	x	x					x			x
[Dey *et al*, 2004]	x					x			x		x		x					x	x		
[Gu *et al*, 2005]	x								x		x								x		
[Haya *et al*, 2004]									x		x	x	x								
[Kwahara *et al*, 2004]	x	x			x				x		x							x	x		x
[Laerhoven, 2002]											x										x
[Schmidt *et al*, 1999]					x				x				x						x		x
[PlaceLab, 2004],[Intille *et al*, 2005]	x			x	x	x	x		x		x	x	x			x		x			
[Cyberkinetics, 2005], [Gasson *et al*, 2004], [Leuthardt *et al*, 2004],			x														x				

Table 7. A comparison of sensor data classifications performed by various researchers and institutions

The comparison shows that a heterogeneous set of sensors is needed for identifying various context descriptors and that no single author has been able to develop sensor data classifiers which completely address the 10 categories of context descriptors shown in Table 7. In most of the cases researchers build a hardware prototype, which can be bulky and take considerable time and resources to develop. Often the work of one researcher would focus on one context descriptor such as in developing sensors to determine a person's physiological states. Amongst authors interested in investigating context awareness, few sensors (location and environmental sensors) are used. Multi sensor use, as shown in Table 7, does not necessarily imply use of a variety of context descriptors. Because of hardware unavailability, few of the sensor modalities which matter to people are investigated. Table 7 shows that video and audio are most exploited. The three example works which use smell, touch and thought sensors are referenced for completeness. Using such sensors in experiments is complicated by the fact that they require specialized sensing equipment.

3.4.4 Rapid programming environments for supporting context aware application development

This work believes that the problems which are part of context aware application development can be reduced by using rapid application programming environments which aid in context aware application development. This section considers a number of existing proposals for rapid prototyping tools which can be used for aiding the development of systems which use sensors and sensor data analysis components. However, these systems are not specifically tailored to context aware application development. They lack support for management of data related to context and do not provide facilities such as context interpretation. In addition, no single system provides all the required facilities.

3.4.4.1 GUI based programming environments

GUI based integrated development environments (IDEs) such as Microsoft's Visual Studio [Microsoft, 2005b] and the NetBeans open source IDE [NetBeans, 2005] are examples of rapid prototyping tools for developing non-context aware applications.

These development environments are suited for developing widget based systems. Gilleade [Gilleade *et al*, 2003] proposes Liquid, a piece of software that acts as a universal interface to sensors which aims to allow the collection and representation of sensor data from a wide range of sensors. It aims to provide a general purpose toolkit for developing sensor based systems to ubiquitous application developers (rather than context aware application developers). This system is limited to sensor development. Dey [Dey *et al*, 2004], MacIntyre [MacIntyre *et al*, 2004] and Freudenthal [Freudenthal *et al*, 2004] also propose graphical programming environments for systems that make use of and analyze data from multiple sensors. Dey's idea is to get users involved in the context interpretation process by using a GUI application to teach a context interpreter how it should recognise user context. MacIntyre's idea is to create a visual application composition environment, based on Macromedia Flash [Macromedia, 2005], which can be used for building sensor based systems rapidly. Freudenthal's idea is to create a visual programming environment which he states is essential for making it easier for students to build sensor based systems. Of all these systems, only Dey's system provides some support for context aware application development.

3.4.4.2 Visual data flow based programming environments

Visual data flow based programming systems such as VIPER [Sanner *et al*, 2002], IBM's data explorer [OpenDX, 2005], Directshow [Microsoft, 2005c], [Davis, 2005] enable building systems using combinations of components which are connected together as a graph. Components process data as it flows through them. VIPER is a general data flow system. IBM's data explorer is used for data visualization. Directshow is used for processing multimedia data. Directshow's graph manager is a graph based modelling tool that can be used to build directed graphs consisting of components called filters which form vertices of a graph. HIVE [Minar *et al*, 1999] also resembles the above systems but does not explicitly support data flow based programming. It is a system that can be used to model the interactions between distributed components called agents, which are contained in cells. Cells, which represent graph nodes, run autonomously and contain additional components called shadows which represent APIs to local resources. Communication between cells is bi-directional.

Data flow systems simplify the building of data centric systems, and GUI versions of the above systems also enable rapid visual programming. For this reason, this work will adapt data flow based programming for its experimental platform as described in chapter 5.

3.4.4.3 3D based programming environments

Visual programming environments that have potential for enabling rapid system building in the future are 3D based programming environments. Barton's [Barton *et al*, 2001] UBIWISE contains a 3D environment where ubiquitous (rather than context aware) devices, such as digital cameras and cellular phones, can be modelled and user interactions with modelled devices can be studied. Barton has produced an example device modelling environment which uses the Quake III 3D game engine. Barton's concept is to make use of existing 3D gaming systems for modelling systems. Another example of a 3D modelling environment is 3D Studio Max [Autodesk, 2005], which can be used to model and script 3D animations.

Even though 3D based programming is important in rapid application development, the provision of a 2D context aware application development tool is useful before a 3D system can be considered.

3.4.4.4 Hardware based prototyping

In hardware based prototyping, small general purpose programmable hardware systems are built separately. A bigger system can then be built by connecting together smaller hardware components [Schmidt, 2002]. This work believes that before attempting hardware based prototyping, a software only approach that enables the building of complex systems out of smaller components is more useful. Such a system can free individuals from limitations imposed by hardware availability. The experimental system which is developed during this work (introduced in chapter 5) integrates this functionality.

3.5 Developing a context aware machine-based memory manager

All the works considered in section 3.3 provide a generic architecture for context aware application development. Although they identify essential parts of context aware architectures, they are not directly useful in constructing a model for a machine-based memory manager. It has also been established in section 3.4 that developing systems which are based on them is difficult. This section describes how these issues can be dealt with in order to develop a context aware machine-based memory manager.

3.5.1 Application centric vs. memory centric model of context awareness

In each work considered in section 3.3, individuals are concerned with providing services and applications to users. They use a few sensors and context descriptors as well as a limited domain context model which applies to the service or application they provide. All the architectures contain an entity called a context provider which is responsible for providing context to applications. Their model of context awareness is *application centric*. In contrast, the machine-based memory manager which this work is interested in developing supports and augments individual's biological memory system. It does so by enabling them to retrieve and use external memory by context. The retrieved external memory fragments can then be used in conjunction with any non-context aware application. This model of context awareness is *memory centric*.

3.5.2 Adapting a model of context awareness for use in this work

Because the works considered in section 3.3 are application centric, they lack the following features which are essential to a memory centric machine-based memory manager which will be developed in this work.

- Sensor diversity

A variety of sensors enable the capture of the various states of people and their environment.

- Context descriptor diversity

A diverse set of context descriptors provide more options for tagging and retrieving external memories.

- A generic context model

A generic context model which can grow and is able to accommodate the multitude of states of people and their environment.

- Long term context descriptor and external memory management

Because the machine-based memory manager does not forget memories it needs to be able to store and organize external memories and context descriptors that accumulate over time. This can be done by automatically organizing external memories and context descriptors as they are captured, rather than capturing and accumulating a mountain of data which then has to be mined for useful external memories.

- Decoupling of applications from context

Besides just decoupling applications from dealing with the physical and context layers of a context aware system, which the works in section 3.3 have done, this work sees as a problem, the creation of applications that understand context. Developing many applications each of which has to understand a particular context is not necessary. Applications can be decoupled from context by using a memory centric context model whereby a personal memory manager enables the delivery of external memories by context to (non-context aware) applications which process them.

Chapter 4 develops a memory centric, context aware model for a machine-based memory manager which addresses the above issues.

3.5.3 A software infrastructure for developing the machine-based memory manager

Section 3.4 showed that it is difficult to construct a context aware system that contains the features described in the previous section. Therefore, this work proposes that it is

important to conceptualise and develop a single software environment which contains facilities that support:

- the design and use of a variety of virtual sensors and real sensor drivers
- the design and use of components that can identify a diverse set of context descriptors
- context modelling
- context aware application prototyping with emphasis on
    - rapid and visual application and simulation construction
    - component reuse

Chapter 5 will conceptualise and implement such a software environment which will then be used in chapter 6 to develop an experimental machine-based memory manager that uses a variety of context descriptors to manage a person's external memories.

4 A model for a machine-based personal memory manager

The consideration of existing memory management research in chapter 2 shows that a model for machine-based memory management is needed. This chapter develops such a model. The model explains machine-based personal memory management by drawing from concepts that are part of the research which is considered in chapters 2 and 3. In the chapter, the aim of the model and its requirements are specified. A number of entities and processes which form the basis of the model are identified. The model is then described in terms of these entities and processes.

4.1 The aim of the model and its requirements

The aim of the model is to provide a simple explanation of how a machine can manage all the external memories of an individual. External memories are past states of people and their environment as perceived by a machine. As stated at the end of chapter 3 (section 3.5.2), the model is memory centric rather than application centric. Therefore, the machine which is described by the model provides a different type of personal memory management than that performed by traditional systems such as PIMs. These systems consist of applications which provide facilities that manage a limited set of external memories such as contacts, calendar items and meetings. There is no reason why personal memory management should be limited to just these few aspects of a person's life. A normal person, everyday, smells various fragrances, hears different sounds, has visual experiences, tastes foods, touches various objects, interacts with different people, experiences a range of feelings and engages in many more activities which are part of life. Therefore, the machine which is described by the model aims to manage any type of external memory.

4.1.1 A machine-based personal memory manager according to the model

A machine-based personal memory manager, according to the model which is developed in this chapter, is an entity that is part of a person's private space. Its purpose is to capture the states of its owner and his environment using artificial sensors and store them as external memories on artificial storage devices. This is done in order to enable its owner to consume (re-experience) his external memories at a future time by describing them using his past states. The machine-based memory manager is required to perform its function throughout the life of its owner and to support and augment his owner's biological memory system. The following comparison distinguishes a machine-based memory manager from traditional memory management systems. It also shows the level of disparity that exists between the two systems.

Traditional memory managers	A machine-based memory manager
• systems such as desktop computers provide hundreds of applications. Data associated with applications is scattered all over the desktop environment. Individuals are required to manually organize their external memories. • external memories accumulate over long periods of use. Finding them is difficult and time consuming. • ignore a large amount of information that describes the states of a person and his environment which are part of a person's day-to-day life. • systems such as PIM devices provide facilities such as address book and contact management enable people to communicate with each other and can contain sensors such as video and audio recorders. However, they have the same problems as desktop computers.	• serves as a memory appliance rather than providing hundreds of applications to people. • is a proactive entity which uses multiple artificial sensors to monitor and capture the states of people and their environment. People do not have to organize their external memories. • allows external memories to be retrieved by describing them with past states of people and their environment. • keeps track of a person's external memories throughout their lives thereby enabling their owner never to forget them. • augments a person's biological memory system by extending it with artificial sensors and subsystems that perform functions which the human memory system cannot.

4.1.2 Requirements of the model

The main problem which is identified at the beginning of this thesis is: *the provision of a machine which can support and augment an individual's biological memory system*. The research that has been considered thus far has led this work to conclude that for a machine to support and augment an individual's biological memory system in day-to-day use, the machine:

1. needs to be an intimate entity which is part of a person's life
2. needs to be able to capture the states of its user and his environment with artificial sensors and store them as external memories
3. needs to provide an effective method for organizing external memories as they are stored
4. needs to facilitate the retrieval of external memories through descriptions of past states of an individual and their environment
5. needs to target specific parts of the human memory system for support and augmentation
6. needs to be able to perform its function over long periods of time which means the amount and type of external memories that are managed is large

The model, therefore, needs to explain how the above requirements can be fulfilled by a machine-based memory manager.

4.1.3 Relating the requirements of the model to research considered in chapters 2 and 3

Requirement 1

The human memory system, for which psychological theories are proposed as shown in chapter 2, is an intimate system. The aim of machine-based memory aid research considered in chapters 2 and 3 is to develop an intimate machine-based entity that supports and augments a person's biological memory.

Requirement 2

The information processing model of memory which is considered in chapter 2 provides a possible method of fulfilling this requirement. If this model is to be used as a base for a machine-based memory manager, then the machine should have a sensing subsystem and different memory subsystems such as short term memory and long-term memory. According to this model stimulus from the environment is acquired using sensors and is stored in short term memory where its usefulness is evaluated. Based on the results of the evaluation, memory is either placed in long-term memory or discarded. However, this work and the works reviewed in chapter 2 on long-term memory management realise that it is more advantageous not to discard memories when using machines to manage memories in order to enable people never to forget any of their external memories.

Requirement 3

The levels of processing model which is part of the information processing model of memory further proposes a method by which stimuli from sensors can be processed to extract features that aid in memory management. It states that stimuli acquired by sensors should be processed in stages, where each stage represents a different semantic level of interpretation of the stimuli, until features that can be used for encoding or decoding memories are identified. Context aware computing provides the concept of context descriptors which this work and the works reviewed in chapter 3 believe are an effective method for tagging and facilitating the retrieval of external memories. Context descriptors are defined in chapter 3 as features extracted from sensor data by context interpreters which help determine context. Context interpreters can use the levels-of-processing approach to identify context descriptors.

Context aware computing also provides the concept of a context model which is used by context aware applications for inference. A machine-based memory manager can make use of a context model which is built using context descriptors for encoding and decoding memories.

Requirement 4

This work believes that context descriptors can be used as an effective method for memory retrieval and thus can fulfil requirement 4. In addition the interface between humans and the machine needs to be considered in relation to memory retrieval and consumption. The research considered in chapters 2 and 3 identifies methods of memory retrieval and consumption which are close to the way humans naturally retrieve and consume memories from their biological system such as brain computer interfaces (BCIs) for thought-based retrieval and virtual displays for private consumption of memories.

Requirement 5

The information processing model, considered in chapter 2, develops short term memory subsystems such as the phonological loop (verbal memory support), the visuo-spatial sketchpad (visual memory support) and the central executive (short term memory coordination) as well as long-term memory subsystems such as procedural (unconsciously accessed memory such as skills) and declarative memory (semantic and episodic memory). Besides visual and verbal short term memory subsystems, the memory manager can also use artificial sensors that enable humans to possess new types of memory subsystems called extra sensory subsystems. Although they identified the need to support the human memory system with machines, machine-based memory aid researchers whose work is considered in chapter 2 and 3, did not explicitly identify the above human memory subsystems and specify how they can be supported and/or augmented.

Requirement 6

Long-term memory management is a feature of the human memory system. It is also the aim of long-term memory research considered in chapter 2. When fulfilling requirement 3, the model needs to take into account long-term memory management to enable a person never to forget any of their memories.

4.1.4 Fulfilling the requirements of the model

As a hypothesis, this work proposes that the requirements of the model can be fulfilled by applying the following principles.

1. Identifying entities and processes which are part of existing models of memory management such as psychological models of memory which are considered in chapter 2 and context aware computing which is considered in chapter 3.
2. Reducing the number of identified entities and processes to form new ones by grouping those entities or processes that perform similar and related tasks together. Fewer entities and processes means a simpler model.
3. Describing the identified processes in terms of the roles of the entities that participate in each process.

The hypothesis will be tested through the construction of a useful model which follows the above principles.

4.2 Entities and processes that form the base of the model

Before presenting the model, entities and processes that can be used as a base for the model need to be identified. This can be done by following the steps which are outlined in section 4.1.4.

4.2.1 Candidate entities and processes

The research considered in chapters 2 and 3 as well as the discussion in the previous section identify a number of entities and processes that this work believes can be incorporated into a model for machine-based memory manager. These include:

Entities

- *environmental stimuli (states)*: states of a person and their environment
- *sensors*: used to capture environmental states
- *external memories*: states of people and their environment as perceived by a machine
- *context descriptors*: features extracted from sensor data useful for tagging external memories
- *context interpreters*: identify context descriptor states from external memories

- *context model*: used for inference by context aware applications
- *the visuo-spatial sketch pad*: short term visual memory processing subsystem
- *the phonological loop*: short term verbal memory processing subsystem
- *extra sensory subsystems*: subsystems that process memories besides visual and verbal
- *the central executive*: an entity that coordinates and directs short term memory
- *procedural and declarative memories*: types of long-term memory

Processes

- *levels of processing*: performing sensor data analysis in a series of steps, each representing a different semantic level of meaning
- *memory capture, encoding, storage, decoding and retrieval*: processes which are undertaken by the human memory system according to the information processing model of memory

4.2.2 Selected entities

After applying steps one and two of the principles which have been specified in section 4.1.4 as necessary for fulfilling the requirements of the model, the following entities are proposed to form the base of the model.

- *life streams (states of a person and his environment)*

A life stream is a simple term which describes the states of people and their environment. The term is not original. Its meaning and use in other contexts is identified in section 4.2.2.1.

- *memory fragments (external memories)*

A memory fragment is a fundamental unit of memory management. Thus it is recognized as a distinct entity. The term memory fragment is preferred over external memory because individual's external memories are managed as a collection of items which are captured as described in section 4.2.2.2 and 4.3.

- *context descriptors*

Context descriptors are a fundamental means of memory management as described in sections 4.2.2.3, 4.3, 4.4 and 4.5, therefore they are recognized as distinct entity.

- *a memory manager*

A memory manager represents a single logical entity that manages all memory fragments including procedural and declarative memories as described in section 4.4, therefore it is identified as a single entity.

- *memory producers (use sensors, context interpreters, a context model)*

Memory producers are entities which produce memory fragments and/or context descriptors as described in sections 4.2.2.5 and 4.3. Step two of the principles specified in section 4.1.4 is applied to create a memory producer entity.

- *memory consumers*

Parts of the machine-based memory manager which (1) support and augment human memory subsystems (such as the central executive, visuo-spatial sketch pad, phonological loop), (2) that provide extra sensory subsystems, and (3) that use memory fragments during context interpretation, are identified as a single entity, called a memory consumer, as they are all users of memory fragments. Sections 4.2.2.6 and 4.5 describe memory consumers and the processes they participate in.

Figure 7 illustrates the model in terms of the above entities. Unlike the information processing model of memory, no distinction is made between short term and long-term memory. So-called short term memory processing subsystems are consumers of memory fragments (memory consumers). The rest of this section describes each of the entities that are in the model.

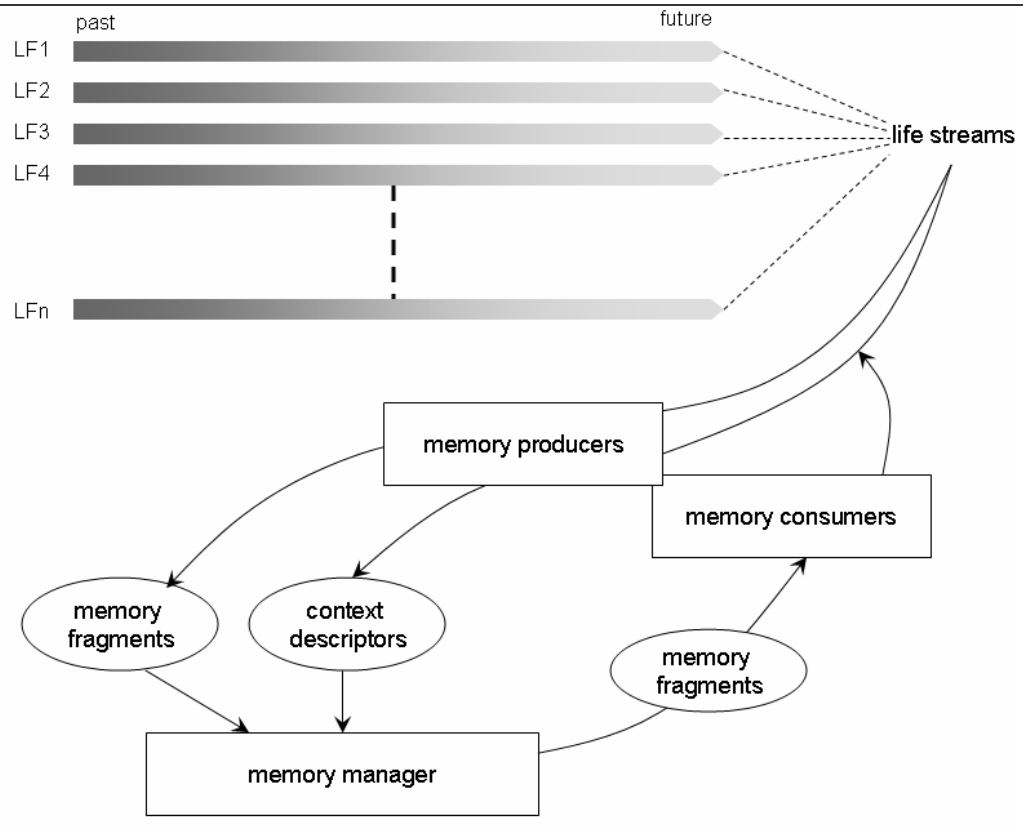


Figure 7. The model in terms of entities which are part of the model

4.2.2.1 Life streams

The term life stream is not original. It has been used by different authors in the past to refer to a subset of the states of people and their environment. Tulving [Tulving, 1985] refers to events as life streams, which he defines as the basic unit of perceived time. Davis [Davis, 1993] refers to streams of video recordings as life streams. Freeman [Freeman, 1997] refers to streams of electronic documents as life streams. In this model, life streams are streams of states of a person and their environment which are continuously flowing in time. Specific examples of some of the life streams that make up a person's life are shown in Figure 8. At any moment in time, a person's life stream contains *n* streams which are denoted as LF1 to LFn in Figure 8. The model assumes that there is no upper limit on the number of streams that a person's life can contain.

Time: a glue that binds life streams

Every person is born at some moment in time and dies at a later moment in time. A person's time stream spans this period. Rekimoto [Rekimoto, 1999, 1999b] states that time can bind life streams together. The very definition of memory as the record of the experience in chapter 2 implies that for memory to form, passage of time is a precondition. Therefore, time has the fundamental property that it can bind all life streams together through the time of attainment of states that constitute life streams.

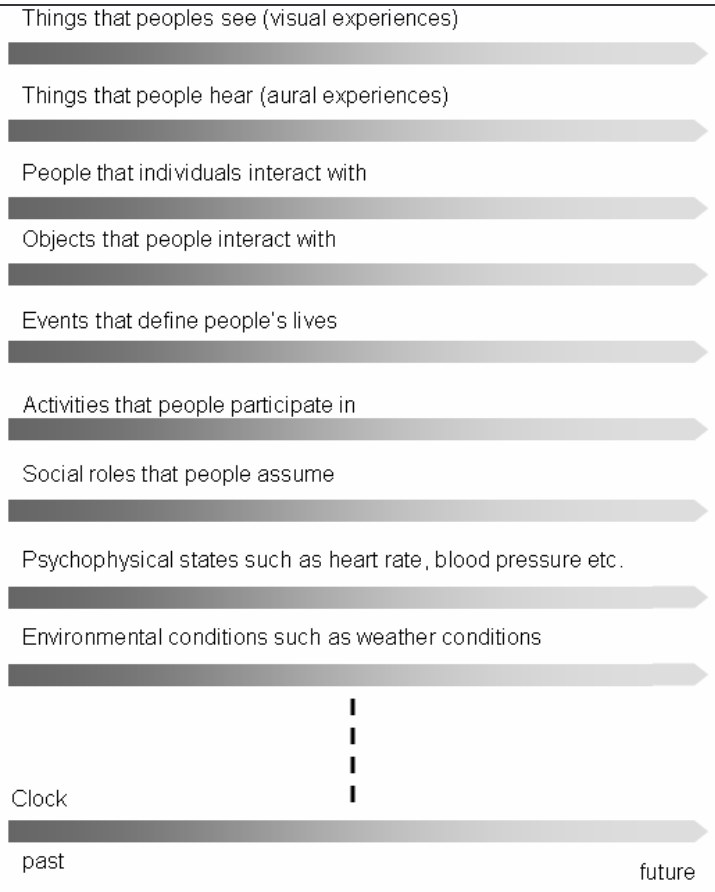


Figure 8. Examples of some of the life streams that make up a person's life

4.2.2.2 Memory fragments

Memory fragments are external memories which are sampled from life streams by artificial sensors and digitized for storage on machine-based memory. As an example, consider visual and auditory memories. When a video recording is made by a person using a digital camera, visual and auditory memory fragments are sampled from a

person's visual and auditory life streams. The concept of sampling of memory fragments from life streams is the same as the sampling of digital representation of data from analogue data. When sampling digital representations of data from its analogue form, the sampling resolution and frequency determine the quality and continuity of the digital data which is sampled. This principle also applies to the sampling of memory fragments (which are digital) from life streams (which are analogue). If a high enough sampling resolution or frequency is not used some states will be missed, which means a machine-based memory will not capture all possible states.

4.2.2.3 Context descriptors

Context descriptors are defined in chapter 3 as classes of states which are members of the context set, where the context set contains states of a person and their environment which can be used to describe memory fragments. Because of their role in describing memory fragments, context descriptors are central to the model. As an example of a context descriptor, consider user location, which is the most widely used and easily identified context descriptor. Location context descriptor states such as *home, office, town, city A, country A* are a group of states that belong to the user location context descriptor class. In the rest of the thesis the term context descriptor or context descriptor state will be used where use of each term is appropriate.

The difference between context descriptors and memory fragments

Both context descriptors and memory fragments are sampled from life streams. This means that context descriptors are also memory fragments. Although this is true, context descriptors have to be distinguished from memory fragments because they have additional properties. Context descriptors are a summary of memory fragments. In the model, context descriptors serve the role of tagging memory fragments and enable associations between different memory fragments which are formed in the same context. The properties of context descriptors are further explained in section 4.4 where the memory encoding and storage processes are described.

4.2.2.4 Memory manager

A memory manager is an entity contained in the model which is responsible for transparently facilitating the encoding, storage, decoding and retrieval of memory fragments. The memory manager provides a logical view to stored memory fragments. Figure 9 illustrates the processes that the memory manager facilitates. These processes are described in section 4.4.

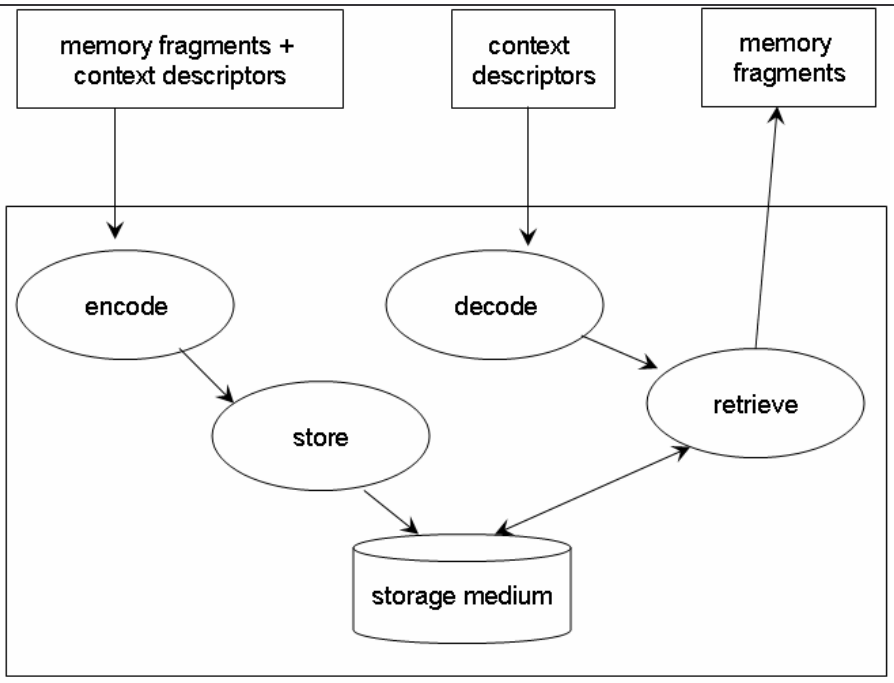


Figure 9. Processes that the memory manager facilitates

4.2.2.5 Memory producers

Memory producers are entities in the model that facilitate the memory capture process. They sample memory fragments from life streams and also identify context descriptors by analysing sampled data. The series of steps through which memory producers perform their functions are described in the memory capture process in section 4.3. In order to identify some context descriptors, past knowledge is necessary. This implies that some memory producers also need to act as memory consumers since they need to consume memory fragments from the memory manager in order to identify new context descriptors. For this reason an overlap exists between memory producers and consumers

as shown in Figure 7 which shows the dual producer/consumer role that is played by some memory producers.

4.2.2.6 Memory consumers

Memory consumers are end users of memory fragments. In the model there are two types of memory consumers: primary and secondary. The primary consumers of memory fragments are entities that provide an interface between humans and the machine-based memory manager. They are identified as primary consumers because the machine-based memory manager is conceptualised to serve humans. Secondary consumers of memory fragments are entities which need to make use of memory fragments for some task. Examples are context interpreters that need to analyze past memory fragments in order to identify new context descriptors. There is a need to differentiate between primary and secondary consumers because humans and entities such as context interpreters consume memory fragments differently. Whereas an efficient communication link can be created between software and hardware components and the memory manager, creating a communication link between humans and the memory manager is more difficult. Memory consumers communicate with the memory manager by altering states in life streams (in the case of primary consumers) or directly when they are part of memory producers (in the case of secondary memory consumers). The interface between humans and primary memory consumers is considered in section 4.5. The interface between secondary consumers and the memory manager is considered in section 4.3.

4.2.3 Selected processes

The processes identified in section 4.2.1 as candidate processes already serve as a grouping that encompasses multiple entities and sub-processes that perform similar and related tasks. They are: (1) memory capture, (2) memory encoding, (3) memory storage, (4) memory decoding and (5) memory retrieval. However, some of these processes are related. Thus, applying step two of the principles specified in section 4.1.4, they can be re-grouped as: (1) memory capture, (2) memory encoding and storage, (3) memory decoding and retrieval. The levels-of-processing approach is a sub-process within

memory capture. Each process has its own requirements which need to be fulfilled in order to meet the overall requirements of the model. The rest of this chapter describes each process in detail.

4.3 Memory capture

A memory capture process is necessary in the model to enable the sampling of memory fragments and identification of context descriptors. The previous section identifies memory producers as entities that facilitate this process. While multiple memory producers can exist, they all follow the same memory capture process. This section describes how memory producers sample memory fragments and identify context descriptors.

4.3.1 The memory capture process

Entities that make up memory producers are shown in Figure 10. These entities are: sensors, memory fragment encoders and context interpreters. They either produce memory fragments or context descriptors. Sensors capture raw data that represents the states of people and their environment. Memory fragment encoders encode this raw sensor data into memory fragments. Context interpreters analyze raw sensor data and produce context descriptors. Outputs of memory producers are then passed to the memory manager.

- *Sensors*

Sensors are responsible for sampling raw data from life streams. In Figure 10, three types of sensors are shown: hardware sensors, virtual sensors and people+hardware based sensors. Hardware based sensors autonomously perform sampling of raw data from life streams. Virtual sensors emulate the operation of hardware based sensors. They can be autonomous or manually controlled. People+hardware based sensors make use of a person’s biological memory system to identify context descriptors or context profiles from life streams.

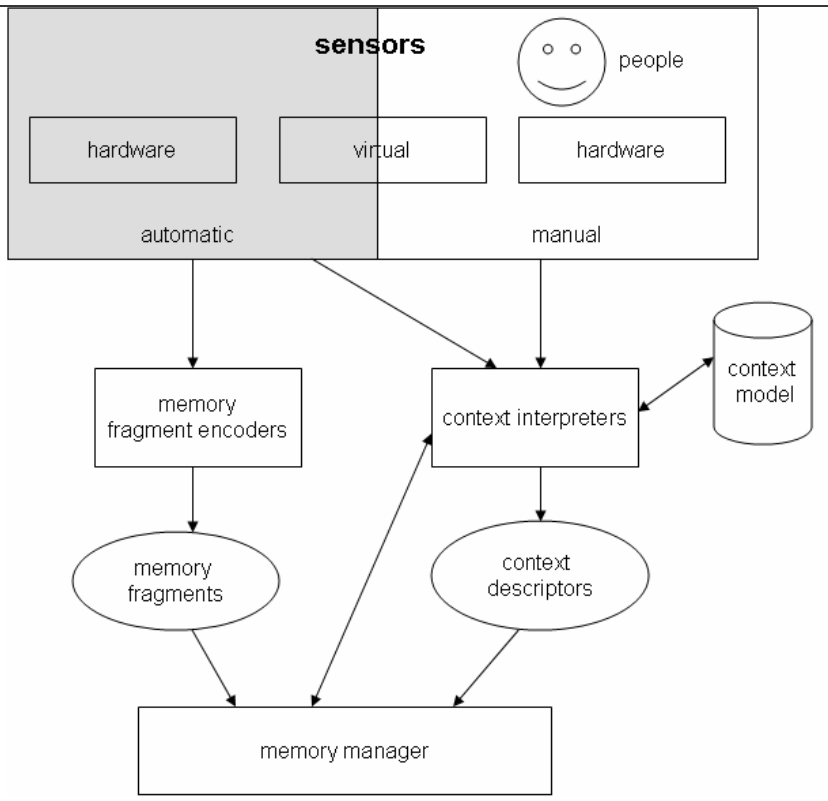


Figure 10. The internal view of memory producers which are entities that facilitate the memory capture process

- *Memory fragment encoders*

A memory fragment encoder is responsible for encoding raw data sampled by sensors into memory fragments. The model accommodates the use of multiple memory fragment encoders though different types of memory fragment encoders.

- *Context model*

A context model is a mapping from real world concepts to collections of context descriptor states. An instance of a mapping is called a context profile. The context model therefore consists of a collection of context profiles. As an example, consider descriptions such as *a red door* or *working at home* which represent context profiles. These context profiles can be broken down to context descriptor states of the form context descriptor (state1, state2,…, stateN) such as object (door), colour (red) and location (home), user mental state (busy), activity (working). Context interpreters that interpret input from humans use the context model to perform mappings from context

profiles to context descriptor states, which eliminates the need for humans to use atomic context descriptor states.

The memory storage mechanism that will be introduced in section 4.4, called a quanta based context aware file system, enables the creation of relationships between context descriptor states automatically as they are identified by memory producers and stored by the memory manager. This feature can be seen as an extension of the context model because it enables a context interpreter to discover relationships that exist between context descriptors. It represents an ontology which is built automatically.

- *Context interpreters*

Context interpreters are one of the components which are identified as being part of context aware architectures in chapter 3. Context interpreters are entities that take as input raw sensor data and/or collections of context descriptors and produce as an output a new type of context descriptor. A context interpreter can make use of a context model to identify new context descriptors. As an example of context interpretation, consider memory fragments which are sampled from a temperature life stream using a temperature sensor. By analysing the raw data sampled by a temperature sensor, a context interpreter which has been created to identify the three distinct states *hot*, *cold* or *mild* would produce one of these states as temperature context descriptors states. A context interpreter can also aggregate and perform inference on collections of context descriptors to produce new context descriptors. A simple method for performing inference on collections of context descriptors is illustrated by the following symbolic representation:

$$Cd_n = Map\ (Cd_1, Cd_2, \ldots, Cd_i)$$

where Cd_n is a new type of context descriptor, Cd_i are input context descriptors, and Map is a function that performs inference and identifies Cd_n.

4.3.2 Memory fragment capture

An artificial sensor that can sample digital data (memory fragments) from life streams is necessary to facilitate memory fragment capture. Several types of artificial sensors exist which can capture the states of people and their environment. The research considered in chapter 3 shows that video and audio memory fragment recorders are the most widely used. Multiple channel analogue to digital converter devices such as those proposed by Laerhoven [Laerhoven, 2002] and Teller [Teller, 2004] can record multiple types of memory fragments at the same time. Virtual sensors can be used to record life streams which are internal to a computer such as application usage.

4.3.3 Context descriptor extraction

Context interpreters are entities that identify context descriptors. The process they follow to identify context descriptors is known as the context descriptor extraction process which is a sub-process within the memory capture process.

4.3.3.1 Methods of context descriptor extraction

The context descriptor extraction process is illustrated in Figure 11. Two methods of extracting context descriptors from raw data sampled by sensors exist. These two methods are:

- bottom up context descriptor extraction

involves two steps: (1) the use of sensors to sample data from life streams and (2) the use of context interpreters to identify context descriptors from raw sensor data.

- top down context descriptor extraction

involves two steps: (1) the use of sensors (input devices) that detect context profiles from humans and (2) the use of context interpreters to identify context descriptors from human input.

Norman [Norman, 1998] describes the attributes of humans as compared to those of machines using terms such as vagueness/precision, quantitativeness/qualitativeness etc. These terms are adopted here to describe the top down and bottom up context descriptor extraction processes.

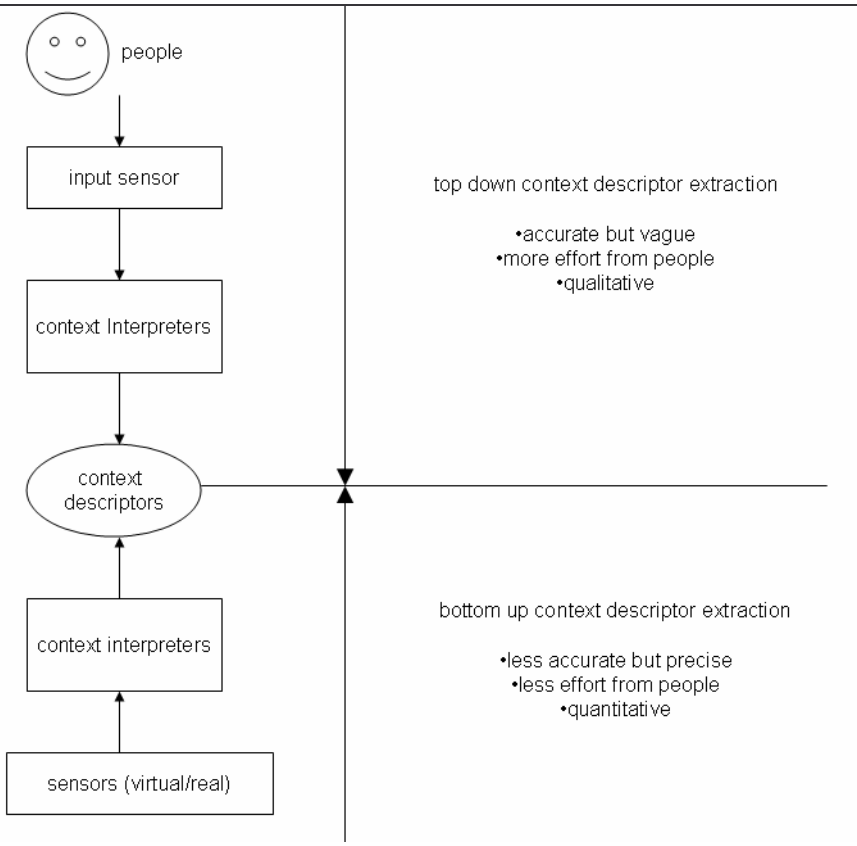


Figure 11. Top down vs. bottom up context descriptor extraction processes

Humans can qualitatively sense context using their biological context interpretation system but their description can be vague (it can contain an element of ambiguity). Context descriptors determined by machines are not qualitative because machines do not possess the same level of intelligence as humans. However, although the accuracy of the absolute values reported by a sensor cannot be guaranteed (due to inherent limitations of sensor hardware), data which is extracted by machines is quantitative. For example, consider the process of determining taste context descriptors. Whereas machines would provide metrics of taste based on chemical analysis, humans would do it on the bases of their previous experience and intellect. They provide qualitative descriptions such as

deliciousness using a reasoning which is entirely based on the human's lifetime experience. The context descriptor extraction process facilitates the extraction of context descriptors which are a combination of qualitative and quantitative descriptions that are provided by people and machines.

4.3.3.2 Context descriptor extraction example

The following examples illustrate the context descriptor extraction process. Consider the extraction of context descriptors from the context profiles *user at home* and *reading book.*

User at home

Bottom up

sensor picks up user location in latitude and longitude coordinates

context interpreter A converts latitude and longitude values to a building identifier

context interpreter B identifies the building as a user's home and produces a user location context descriptor state which in this case is *user location (home)*

Top down

user indicates he is at home by pressing a button

sensor picks up that a button is pressed

context interpreter C maps the button ID to a context profile

context interpreter D maps the context profile to context descriptor states which in this case is *user location (home)*

Reading book

Bottom up

sensor picks up an object that a user is gazing at

context interpreter A identifies that user has being gazing at the object for a period of time by producing the context descriptor state *user eyes (gazing)*

context interpreter B identifies the object and produces the context descriptor state *object (book)*

Top down

user indicates that he is reading a book by issuing the speech command *reading book*

sensor picks up speech

context interpreter C converts speech to a context profile description *reading book*

context Interpreter D maps the context profile to context descriptor states which in this case are *user eyes (gazing) object (book)*

4.3.3.3 Context descriptor extraction through levels of processing

Using the levels-of-processing approach reasoning can be performed at different levels by cascading and interconnecting context interpreters as shown in Figure 12. Using this approach, context descriptors states which have meanings at different semantic levels can be identified.

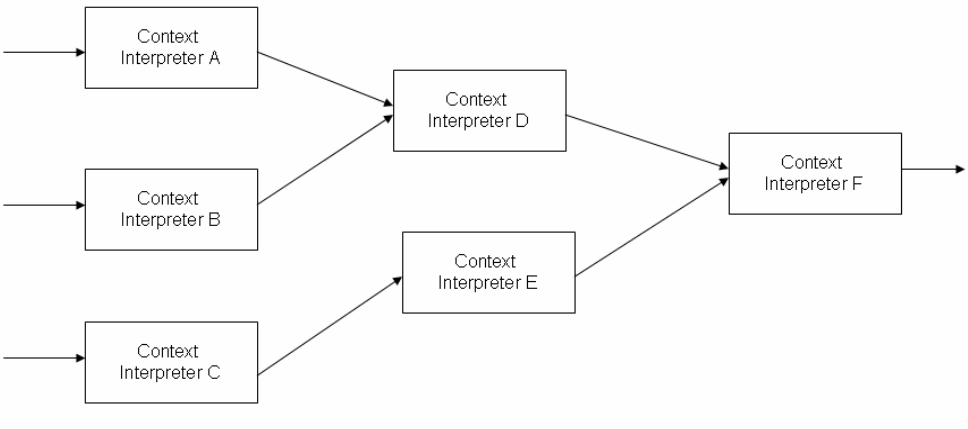


Figure 12. Levels of processing for context descriptor extraction

The number and configuration of context interpreters needed would be dependent on the semantic level of the context descriptor which is being identified. Figure 13 provides a specific example of the levels of processing approach to context descriptor extraction.

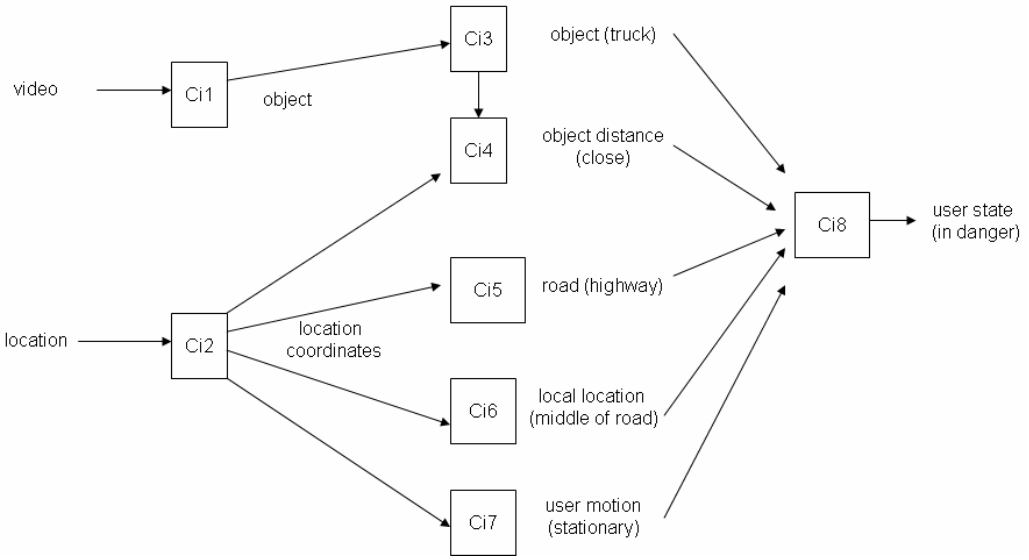


Figure 13. A specific example of levels of processing approach for context descriptor extraction

4.3.4 Manual vs. automatic memory capture

It is desirable that the memory capture process should be fully automatic and make use of real sensors. An example of a fully automatic and autonomous memory capture system is the human biological memory capture system composed of the senses and the nervous system. Machines have yet to match this memory capture system. The model, presented in this chapter, is a step towards a fully automatic, transparent and autonomous memory capturing system. In the model, the memory capture process can be machine or human controlled. Figure 10 shows this by categorizing sensors into manual (human controlled and virtual) as well as automatic (autonomous, hardware based and virtual).

In an ideal world where there are fully autonomous sensors of every kind available, there is no need to distinguish between manual and automatic memory capture. However, in a world where hardware limitations exist, a manually controlled or virtual sensor driven memory capture process is necessary. Making use of virtual sensors (simulation of sensing hardware) enables the development of a memory capture process which is not limited by real sensor availability. Using virtual sensors to develop a memory capture process can also lead to the identification and development of new types of sensors which at some future date can be converted into real, autonomous hardware. Thus, in order to realize a facility that will enable the development of autonomous and fully automatic

sensing systems, the model specifically encourages the use of virtual sensors as well as manual and automatic memory capture processes.

4.4 Memory encoding and storage

An effective memory encoding and storage scheme is necessary to enable a machine-based memory manager to keep track of memory fragments which are produced by memory producers. Context descriptors are introduced into the model in the previous section, as entities that can be used to enable the memory encoding process and the memory manager is identified as an entity that can encode memory fragments using context descriptors and store them internally. This section presents a method that this work has developed to enable the memory manager to encode and store memory fragments using context descriptors.

4.4.1 Requirements for a memory encoding and storage scheme

To fulfil the memory management requirements which are set by the model as specified at the beginning of this chapter in section 4.1.2, three requirements need to be met by the encoding and storage scheme used.

1. The encoding scheme should enable the addressing of memory fragments using context descriptor states. This enables memory consumers to retrieve memory fragments by describing them with context descriptor states.
2. A relationship needs to be maintained between memory fragments and context descriptor states which are identified during the same instant in time when encoding memories. This enables all memory fragments and context descriptor states which are formed in the same context to be associated to each other.
3. The storage scheme used needs to be able to store memory fragments over a long period of time. No memory fragments should be discarded.

4.4.2 The context aware file system: a proposal for a memory encoding and storage scheme

In order to meet the memory encoding and storage requirements which are specified in the previous section, this work has developed the concept of a context aware file system. At the end of chapter 3, the need for a storage mechanism that can store and automatically organize context descriptors and memory fragments which accumulate over long periods of time was identified. None of the authors whose works are considered in chapter 2 and 3, have conceptualised a similar storage mechanism. Early proposals for context aware file systems developed by Hess [Hess, 2003] and Khungar [Khungar *et al*, 2004, 2005] do not fully realise the potential of context descriptors in helping to manage all of a person's memories. Like the context aware architectures which are considered in chapter 3, they are application centric rather than memory centric. Their proposed context aware file systems are also implementation specific (closely related to the hardware they use).

General design considerations

The context aware file system (CAFS) is to be used to organize memory fragments by context. This is conceptually similar to a database which is used to organize data. Beynon-Davies [Beynon-Davies, 1992] asserts that the organizational property of every database implies it provides data abstraction (logical representation of data is separate from physical implementation concerns), data integration (no unnecessary data redundancy), data sharing, data independence (data organization is transparent to users of the data), data integrity (accurate) and data security (access control). The design of the context aware file system (CAFS), described in this section, is concerned with abstraction and integration with particular focus on enabling context based and lifelong memory management. Data sharing and independence are addressed by the memory manager entity. Data integrity and security concerns are not addressed.

The context aware file system

The context aware file system (CAFS) which is developed in this work is shown in Figure 14. Streams of context descriptor states and memory fragments are absorbed by

the CAFS which consists of two storage layers: a context descriptor layer and a quanta file system layer.

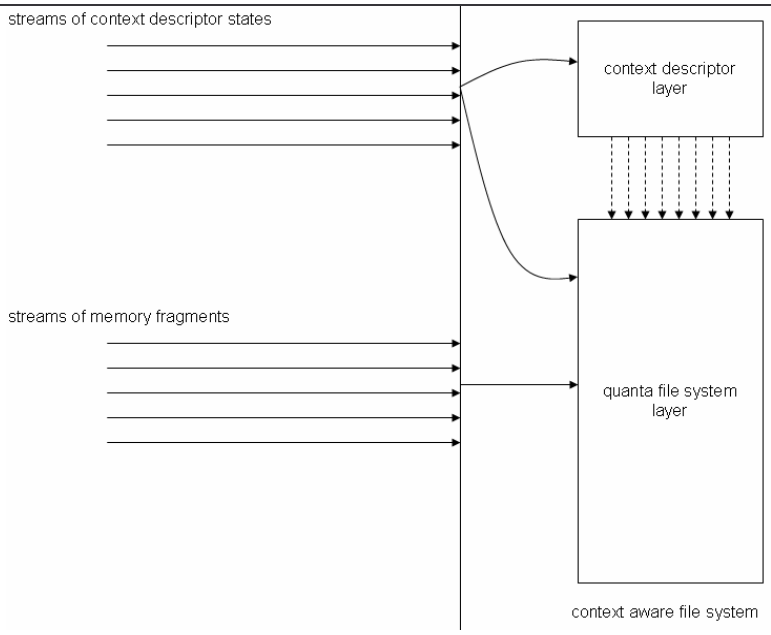


Figure 14. The proposed context aware file system

4.4.2.1 Context descriptor layer

The context descriptor layer fulfils requirement 1 which is specified in section 4.4.1. It contains a collection of context descriptor states that point to collections of memory fragments thereby playing the role of tagging memory fragments. The context descriptor layer of the CAFS stores context descriptor states as well as time stamps that indicate the time of identification of context descriptor states. By examining a particular context descriptor layer it is possible to determine all the times that a particular state of a context descriptor is attained. The relationship between context descriptor states and quanta is illustrated in Figure 15.

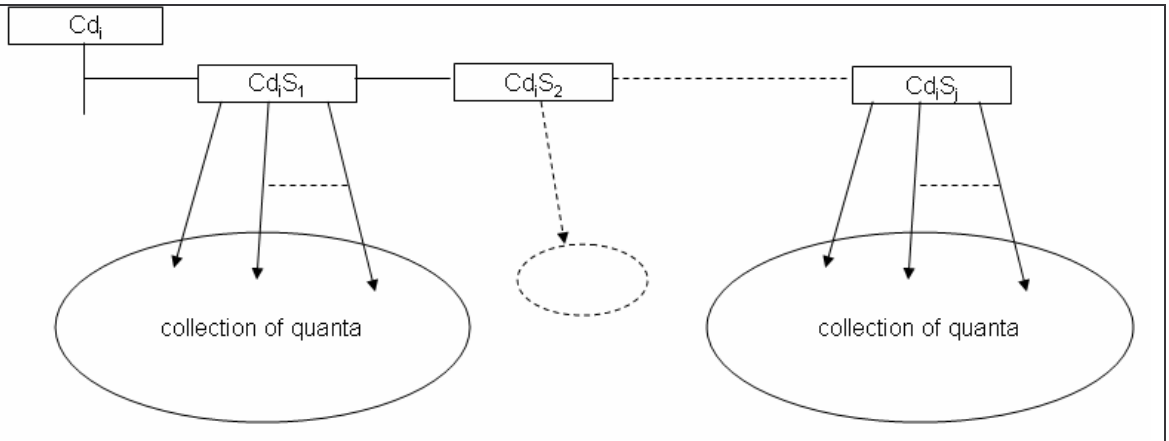


Figure 15. The relationship between context descriptor states and quanta (Cd_i is the ith context descriptor, Cd_iS_j is the jth state of Cd_i)

4.4.2.2 Quanta file system layer

The quanta file system layer fulfils requirements 2 and 3 which are specified in section 4.4.1. It consists of a storage medium which is divided into temporal quanta. Inside each quantum context descriptor states (a subset of those stored in the context descriptor layer which apply to this quantum) and memory fragments are stored. Jensen [Jensen, 2005] defines a quantum as a *spatial quantum*, the shortest distance of space supported by a storage system. In the quanta file system developed in this work a quantum in addition to representing a spatial quantum also represents a *temporal quantum* which can be defined as a small window into a temporally ordered stream of memory fragments. If one could look inside a quantum, one would see memories which are formed during the time period that a quantum represents.

The advantage of the quanta concept is that it allows memory fragments from multiple life streams and different types of context descriptor states which need to be temporally and spatially fused, to be conveniently managed. Quanta represent logical units which can be used to manipulate collections of memory fragments. Figure 18 shows the view inside a quantum which shows memory fragments $m_1, m_2, \ldots, m_j$ and three context descriptor states Cd_1S_3, Cd_2S_6 and Cd_iS_i. The fact that these three context descriptor states are present in a quantum means that the memory fragments m_1 to m_j are formed in the context described by the three context descriptor states. By referring back to the context

descriptor layer it is possible to associate the memory fragments in this quantum with other memory fragments that the context descriptors point to. Hence requirement 2 in section 4.4.1 is met.

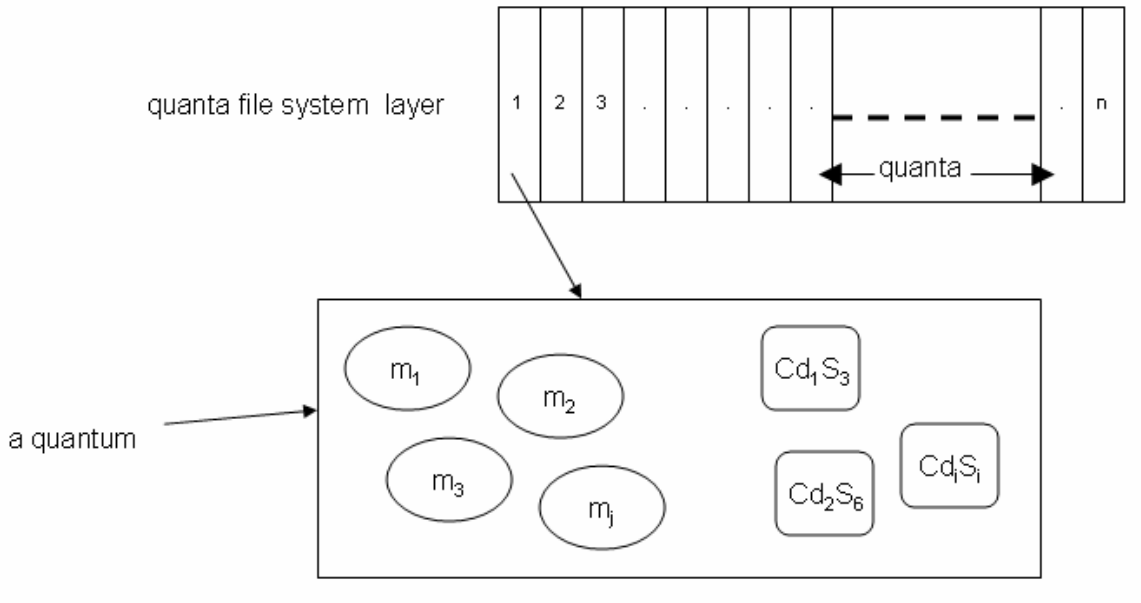


Figure 16. The view inside a quantum

4.4.2.3 Addressing memory fragments in the context aware file system

Two methods of addressing memory fragments can be used in the CAFS: (1) addressing memory fragments by time of formation and (2) addressing memory fragments by context.

Addressing memory fragments by time of formation

If the time period during which a collection of memory fragments are formed is known, as happens when retrieving memories using time intervals, then the context descriptor layer can be bypassed and the quanta layer accessed directly. Locating an address of a quantum is a straightforward operation. The address can be determined using a time stamp which acts as an index into quanta. The calculation of the address is dependent on the temporal size of a quantum which is in use by a particular implementation. Assuming that each quantum is 10 minutes in length, the address of a quantum is calculated by extracting the minute's part of a time stamp and rounding it to the nearest 10^{th} minute.

Addressing memory fragments by context

Memory fragments can be addressed by specifying their context of formation using context descriptor states. In this case, the context descriptor layer contains addresses of quanta which contain wanted memory fragments since it contains a mapping of context descriptor states to quanta. An addressing system that appends a context descriptor and its state such as Cd_iS_i can be used to access the times during which a particular state is attained as well as identify the quanta in which associated memory fragments are stored as described in the previous section.

4.5 Memory decoding and retrieval

The previous section explained how the memory manager encodes and stores memory fragments in order to facilitate their retrieval. In the process, addressing of memory fragments in the CAFS by the memory manager prior to retrieval is described. This section considers memory decoding and retrieval from the perspective of memory consumers.

4.5.1 Requirements of memory decoding and retrieval

Two types of memory consumers are part of the model: primary consumers (present memory fragments to humans) and secondary consumers (e.g. context interpreters). Both consumers need to be able to retrieve memory fragments using the following methods.

- Through context descriptors: description of wanted memory fragments by specifying their context.
- Through memory fragments: presenting memory fragments which are similar to wanted memory fragments (in this case context descriptors would need to be extracted from supplied memory fragments as described in section 4.3).

Methods of handling requests from memory consumers which suit both types of memory consumers also need to be provided. Once memory fragments are retrieved they need to be presented differently for each type of memory consumer.

4.5.2 The memory decoding and retrieval process

Both context descriptors and memory fragments originate from life streams as described at the beginning of this chapter (section 4.2). In the model, memory producers play the role of sampling memory fragments from life streams and identifying context descriptors. Thus memory producers are responsible for decoding requests from memory consumers and the memory manager is responsible for retrieving memories from its storage as described in section 4.4. The memory decoding and retrieval process is illustrated in Figure 19. When memory consumers are interested in memory fragments they alter environmental states in life streams which indicate memory retrieval requests (queries) from consumers. These changes are detected by memory producers which produce context descriptors that can be used for memory retrieval by the memory manager. Memory fragments which are retrieved by the memory manager are presented to humans by primary memory consumers or used by secondary memory consumers.

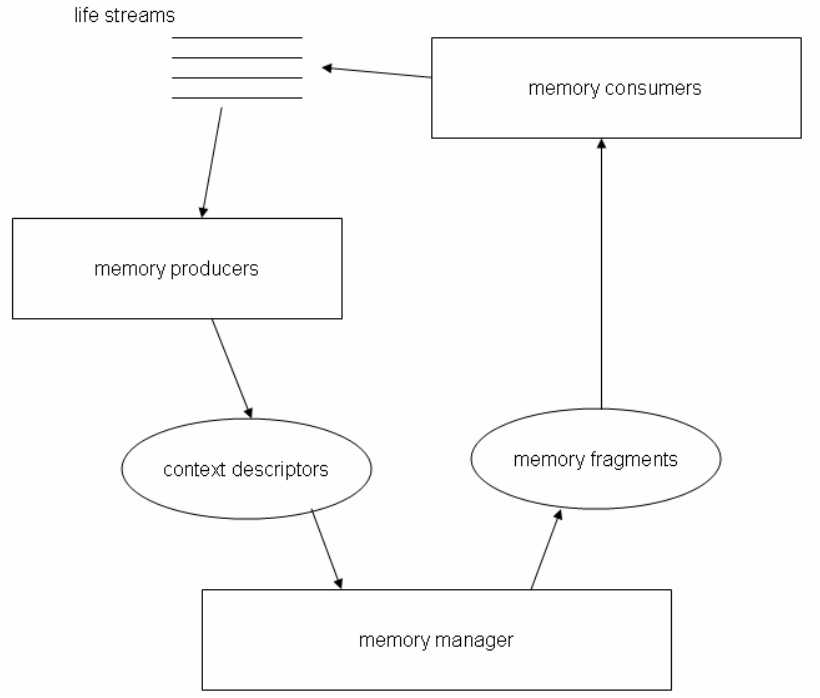


Figure 17. The memory decoding and retrieval process

4.5.3 Effective methods of handling queries for memory fragments

The model contains two types of memory consumers: humans and context interpreters. A query handling method which suits each consumer is necessary. Handling queries from context interpreters is straightforward, since context interpreters are software components. However, the same is not true for human consumers. This is because a human-computer interface problem exists. Naturally, humans use thought based memory retrieval and consumption. Thus, a machine-based memory manager needs a query handling mechanism which is as close to thought based memory retrieval as possible.

Using the concepts introduced in chapters 2 and 3 it is possible to describe how thought based memory retrieval might occur in the future. Consider sensors that can read a person's thoughts or sensors that can directly be wired up to a person's nervous system. Such interfaces are known as BCIs (Brain Computer Interfaces). Early implementations of BCIs already exist. Examples are Warwick's [Warwick, 2005] nervous system implant and Cyberkinetics' [Cyberkinetics, 2005] brain implant. Using BCIs, memory producers can monitor thought life streams and identify context descriptor states that the memory manager can use to decode and retrieve memory fragments. A BCI interface is particularly important in supporting or augmenting human memory subsystems such as the phonological loop and visuo-spatial sketchpad.

In the absence of BCIs, two other possible methods of handling requests for memory fragments are (1) verbal requests and (2) non-verbal requests. Verbal requests which consists of context descriptor states such as those issued through speech are a natural form of communication for humans. Thus they are an effective means of communication. Non-verbal requests that originate from input devices such as a remote control are similar to BCIs, except that humans would first have to think of a code which then has to be mapped to context descriptor states.

4.5.4 Effective methods for presenting memory fragments

Primary memory consumers need to present memory fragments in a manner suitable for human consumption. Humans can request the retrieval of memory fragments for the following reasons.

- *Re-experiencing memory fragments*

In such a case, appropriate applications and presentation devices should be used to enable people to re-experience their memories as close to the original experience as possible. Memory re-experiencing platforms such as Holodecks featured on the science fiction show Star Trek are a possibility for the distant future. Virtual reality based memory re-experience platforms such as those that use head mounted displays (HMDs) [Bungert, 2005] are another option for re-experiencing visual and aural memory fragments. A memory re-experience platform that does not require too much user attention or that which can be used in any environment, without interfering with a user's current activity, would be preferable. Augmented reality memory re-experiencing platforms such as see-through eye mounted displays [MicroOptical, 2003], [TekGear, 2005] that enable the display of memories mixed with real light rays which people perceive can be used for private memory consumption as well as augmentation of the vision. Mann's EyeTap [Mann, 2004] is a concept that aims to achieve this. In the absence of the memory re-experience platforms discussed thus far a simple colour display terminal such as an LCD display or a smaller PDA sized display combined with appropriate software would be sufficient. To enable the re-experience of memories other than visual ones, a platform suited to a particular sensor modality can be used. For example for smell: a smell synthesizing machine [Wyszynskia *et al*, 2005]; for taste: a foodstuff synthesizing machine [Toko, 1998].

- *Sharing retrieved memory fragments*

It is part of human nature to share memories. Primary memory consumers should enable people to share their memory fragments with other people. For example, through the creation of portable packages that can be passed to people or by beaming them directly to

other people's machine-based memory managers. Offering other individuals a controlled view of memory fragments which are managed by a machine-based memory manager through a direct link is another option. Portable package creation software and wireless data transfer mechanisms are ubiquitous in today's computing environment. When integrating them into the memory manager, their operation should be seamless.

5 A software environment for prototyping context aware systems

Realising personal memory management which is based on the model presented in chapter 4 requires a complex combination of software and hardware components. The reason for this is that the model explains the workings of a machine-based personal memory manager which makes use of multiple sensors, entities that analyze sensor data and entities that act as an interface between the memory manager and humans. At the end of chapter 3, it is identified that a software infrastructure which supports the development of such a system is necessary. Furthermore, this work believes that the software infrastructure, once developed, can also be used to support research into future context aware systems, rather than just being a once off tool for constructing an experimental machine-based memory manager. As a result, in this chapter, the software infrastructure is referred to as a software environment which can be used for prototyping context aware systems.

In this chapter, the requirements of the software environment are specified and the facilities which it should provide are identified. As a proof of concept, the design and implementation of such a software environment is presented with a focus on how the requirements of the software environment are met. The chapter ends by presenting examples of applications which are modelled in the software environment.

5.1 The software environment concept

The software environment is a concept for a single system development environment containing all the facilities needed to enable individuals to develop and conduct research into context aware systems [Tsegaye *et al*, 2005]. In developing the software environment concept this work aims to:

1. make it easier for individuals to construct experimental context aware systems

2. provide a software platform that enables a standard experimental platform to be used when constructing and experimenting with context aware systems

5.1.1 Why the software environment concept is important

The software environment concept is necessary because context aware systems are difficult to construct. This is established in chapter 3 where three architectural layers which are common to all context aware systems are identified:

1. the sensor layer: multiple sensing hardware and sensor hardware drivers
2. the context layer: multiple context interpreters and a context model
3. the application layer: applications that interact with the context layer and serve users

When building a context aware system, all of the above architectural layers need to be provided. Individuals, whose works are considered in chapter 3, build a context aware system through a number of steps.

1. Perform a search in order to find existing solutions. For example: existing sensors, sensor drivers, context interpreters, context model building software, and application building software.
2. If existing solutions are not found, propose and build new ones. Thus if any of system components listed in (1) are not found then they have to be designed, implemented and tested.
3. If existing solutions are found then they often have to be modified to fit an individual's architecture.
4. Identify a suitable programming language and use it to implement an architecture, then applications that use the architecture.

The software environment can simplify the above processes by integrating facilities needed to build a context aware system into a single environment which enables rapid context aware application development through visual programming, component reuse and convenient extensibility. This would enable individuals to focus on designing and

running experiments, rather than having to write, compile and debug components which are needed to construct a complex context aware system or spending time in adapting existing implementations of context aware architectures.

5.1.2 The need for the software environment to be a hybrid system

The previous section asserted the necessity of a software environment which enables rapid programming with visual programming as a feature. However, the software environment needs to be a hybrid between a visual programming environment and a traditional programming environment where system construction is performed though scripting or coding using a 3rd generation programming language. There are two reasons for this.

1. The software environment needs to be constructed and be extensible through new or existing components which have to be integrated to work together using traditional 3rd generation programming languages. This is because system building through the use of third generation languages is currently the norm.
2. Individuals can argue that visual programming cannot be used to develop every possible kind of context aware application. In addition, individuals can provide subjective arguments why visual programming is not necessary.

Thus, by being a hybrid software environment, both approaches to context aware application development can be supported.

5.1.3 Facilities that the software environment is required to provide

The facilities which the software environment is required to provide are directly related to the features of context aware applications which are identified in section 5.1.1. The software environment should provide facilities which support the building of a sensor layer, context layer and application layer. A fourth layer which is often forgotten in software development is the user. A context aware system is constructed to ultimately serve a user [Jameson, 2001]. Therefore the software environment should also provide a

facility for modelling the user. The list of facilities which are important for the software environment to provide can be put into four categories:

1. sensor management
    - makes it easy to integrate, test and use new sensor drivers as well as integrate existing ones
    - makes it easy to use real sensor drivers in applications (either to integrate existing ones or develop new ones)
    - provides the option of using virtual sensors when no real sensors are available
2. context management
    - makes it easy to design, test and use new context interpreters as well as integrate existing ones
    - makes it easier to use context interpreters
    - enables the building, testing and use of context models
3. user modelling
    - enables the modelling of the states of users and their environment using a context model
4. application and simulation modelling
    - makes it easy to construct and run simulations and applications
    - enables simulations involving context aware applications and user models to be performed
    - enables rapid application development through visual programming and component reuse

5.1.4 The software environment in relation to this work

In the context of this work, the software should support the construction of all the entities which are part of the model presented in chapter 3. These entities are memory producers (sensor layer and context layer), the memory manager (context layer) and memory consumers (application layer). In addition, it should enable the construction of an

experimental machine-based memory manager through a visual programming environment which stresses rapid system construction and component re-use.

5.2 The design of the software environment

This work is interested in developing a proof of concept implementation of the software environment, which is described in the previous section, for use in constructing an experimental machine-based memory manager. This section presents the design of the software environment.

5.2.1 Existing system building approaches

Prior to designing the software environment, it is important to identify existing system building approaches. This section looks at four types of programming environments: (1) Third generation language based programming environments, (2) Visual integrated development environments, (3) Visual graph and data flow based programming environments and (4) Visual modelling environments.

5.2.1.1 Third generation language based programming environments

In third generation based programming environments a system is coded entirely using a third generation programming language such as C, Java and C#. The programming is often performed using a text editor. After a program is written, it has to be compiled to produce machine code before it is run. The program also needs to be debugged to get rid of errors introduced by humans during system coding.

5.2.1.2 Visual integrated development environments

Visual integrated development environments such as Microsoft Visual Studio [Microsoft, 2005b], Eclipse [Eclipse, 2005] and NetBeans [NetBeans, 2005] IDEs provide visual programming interfaces but these environments still require a lot of code writing using a 3rd generation programming language, compilation and debugging. They support user interface programming through a visual programming paradigm but fall short of supporting entire system building using a visual programming approach.

5.2.1.3 Visual graph and data flow based programming environments

Visual graph and data flow based system modelling environments such as Sanner's VIPER [Sanner *et al*, 2002], IBM's data explorer [OpenDX, 2005], Minar's HIVE [Minar *et al*, 1999] and Directshow [Microsoft, 2005c], [Davis, 2005] enable systems to be built using the following approach.

1. Build system components once using the development approaches described in section 5.2.1.1 (non-visual coding) or 5.2.1.2 (building systems with GUIs).
2. Assemble pre-built components into a complex system.

5.2.1.4 Visual modelling environments

Modelling environments such as 3D Studio Max [Autodesk, 2005] and Macromedia Flash [Macromedia, 2005] enable individuals to create systems by altering the properties of pre-built binary components. The models which are produced by such modelling environments are used in different applications. 3D studio max, for example, enables 3D model creation for applications that make use of 3D animations. Flash is used to produce interactive applications.

5.2.2 An architecture for the software environment

In order to satisfy the requirement of being a simple tool for system construction, the software environment needs to be an environment which primarily enables building systems as described in section 5.2.1.3 and 5.2.1.4 but also supports the system building approaches described in section 5.2.1.1 and 5.2.1.2. Thus, the architecture which is shown in Figure 18 is proposed for the software environment. It consists of black boxes (called data sources, data channels and data sinks), GUI views, a resource manager and a system API. Black boxes enable system building as described in section 5.2.1.3 (graph and data flow based system programming). GUI views enable system building as described in sections 5.2.1.3 and 5.2.1.4 (GUI based modelling). The resource manager and system API enable system building as described in section 5.2.1.1 (programming using a 3rd generation language). However, ideally the only part of the software

environment used by individuals that are interested in constructing experimental systems should be GUI views.

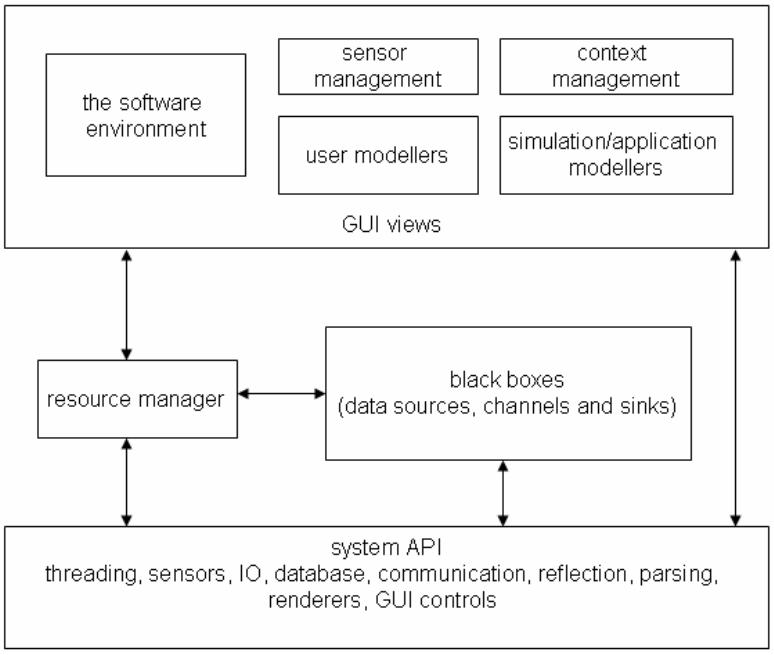


Figure 18. The software environment architecture

5.2.2.1 Black boxes (data sources, data channels and data sinks)

Black boxes are included in the architecture to enable individuals to construct complex systems using small building blocks. Three types of black boxes are proposed in the architecture: data sources, data sinks and data channels. In order to build a complex system in the software environment, larger systems first need to be broken down into one or more of these three components. They encapsulate the various components which are part of context aware systems. Data sources are components in a context aware system, which produce data. Examples of data sources are physical layer components such as sensors. Data channels are components in a context aware system that both consume and produce data. A data channel performs a similar task to a transformation function. Inputs to the function are modified by the transformation operation which is performed by a particular data channel. Examples of data channels are context layer and application layer components. Data sinks are components in a context aware system which consume data.

Examples of data sinks are application layer components such as those that present information to people.

5.2.2.2 GUI views

GUI views are components that present a GUI to each of the facilities offered by the software environment which are described in section 5.1.3. The software environment itself requires a GUI view which enables individuals to navigate through each of the facilities that it provides. GUI views are necessary to make it easy for individuals to make use of each facility provided by the software environment. The concept of a GUI view is also necessary to make it easy to extend the software environment in the future as described in section 5.1.3.

5.2.2.3 Resource manager

The resource manager facilitates the dynamic loading of components which are part of the software environment at run time. For example, it dynamically loads data channels, data sinks, data sources and GUI views and makes them available for use in the software environment. It provides transparent access to sources of data such as a database. Even though use of the resource manager is advantageous, it is not compulsory. This accommodates black boxes that perform their functions using mechanisms which are independent of the general system architecture.

5.2.2.4 System API

The software environment is targeted to be run on a general purpose computer which contains an operating system that makes available low level APIs for access to the computer's hardware. The system API component of this architecture is essential for enabling the resource manager, black boxes and GUI views to make use of low level operating system APIs through a uniform and simplified interface. It provides interfaces for:

- threading: to enable multiple components which make up the software environment to run at the same time

- reflection: to enable persistence for objects which are used in the software environment
- communication: to enable system components to exchange data and objects. For example, for serialization of objects over a network
- parsing: to provide a standard method for formatting streams of data such as streams of context descriptors
- rendering: to enable the presentation of sensor data to individuals such as video and audio using a computer's output devices
- database: to enable access to a database such as a SQL database
- IO: for access to IO devices on a computer
- GUI controls: a set of standard, reusable GUI components for use in GUI views
- sensor access: to enable low level access to sensors which are connected to a system

Providing interfaces to all of the above facilities is necessary because low level operating system APIs on their own are not convenient enough for use in the software environment.

5.2.3 Extending the software environment

The software environment can be extended in three ways.

1. Through new GUI views: if individuals need to add more facilities to the software environment besides those shown in Figure 18, they can do so by implementing new GUI views using the software environment's GUI view API, Resource manager API and system API. This type of extension does not require rebuilding the software environment. It only requires building a new view and adding it to the software environment's GUI view collection.
2. Through new black boxes: if individuals require new types of sensors, context interpreters or applications that present information to users they can extend the software environment by implementing new black boxes using the software environment's black box API, resource manager API and system API. This type of extension does not require rebuilding the software environment. It only requires

building the new black box and adding it to the software environment's collection of black boxes.

3. Through the system API: if individuals require more functionality than is provided by the system API, they can extend it. This type of extension requires rebuilding the software environment.

Examples which show how the software environment can be extended through methods (1) and (2) are presented in Appendix D.

5.3 The software environment implementation

A proof of concept implementation of the software environment has been developed during the course of this work. This section presents the facilities of the software environment which are implemented with the focus on how they met the requirements that are specified in section 5.1.3.

5.3.1 Black box API

The software environment makes it easy to construct complex systems by including the concept of black boxes in its architecture. Examples that demonstrated this feature are provided in section 5.4 and chapter 6. Three types of black boxes are proposed in the architecture (section 5.2.2): data sources, data channels and data sinks. In order to enable individuals to implement these black boxes, the software environment provides an API which consists of three base classes, one for each type of black box. The API specification reflects the properties of each type of black box which are described in section 5.2.2. Data sources produce data autonomously and push it through a *PutToSinks()* method to all sinks attached to them, data channels transform data through a *Transform()* method and data sinks consume data through a *PutData()* method.

- To implement a data source, a programmer needs to use a *DataSource* base class and override the virtual methods *Load(), UnLoad()* and *GetSample()*. To push data to data sinks the base class's *PutToSinks()* method needs to be called.

- To implement a data channel, a programmer needs to use a *DataChannel* base class and override the virtual methods *Load(), UnLoad(), Transfrom(), GetSample()* and *CanConsume()*. To push data to data sinks the base class's *PutToSinks()* method needs to be called.
- To implement a data sink, a programmer needs to use a *DataSink* base class and override the virtual methods *Load(), UnLoad(), PutData()* and *CanConsume()*.

The *Load()* and *UnLoad()* methods should perform initialization and clean up operations. The *GetSample()* method should provide a sample of the type of data a data source or data channel can produce. The *CanConsume()* method should test if a particular data channel or data sink can process a given sample of data. In addition, each base class contains a *Description* attribute which describes what a black box does. C# code samples which illustrate the implementation of black boxes are presented in Appendix D. The operation of the data flow system is further described in section 5.3.6.

5.3.2 The software environment view

The software environment is represented by the GUI view which is shown in Figure 19. The GUI enables individuals to browse the facilities that the software environment provides. Each facility is also represented by one or more GUI views under the following categories: sensor management views, context management views, user modeller views and application/simulation modeller views as shown in Figure 19.

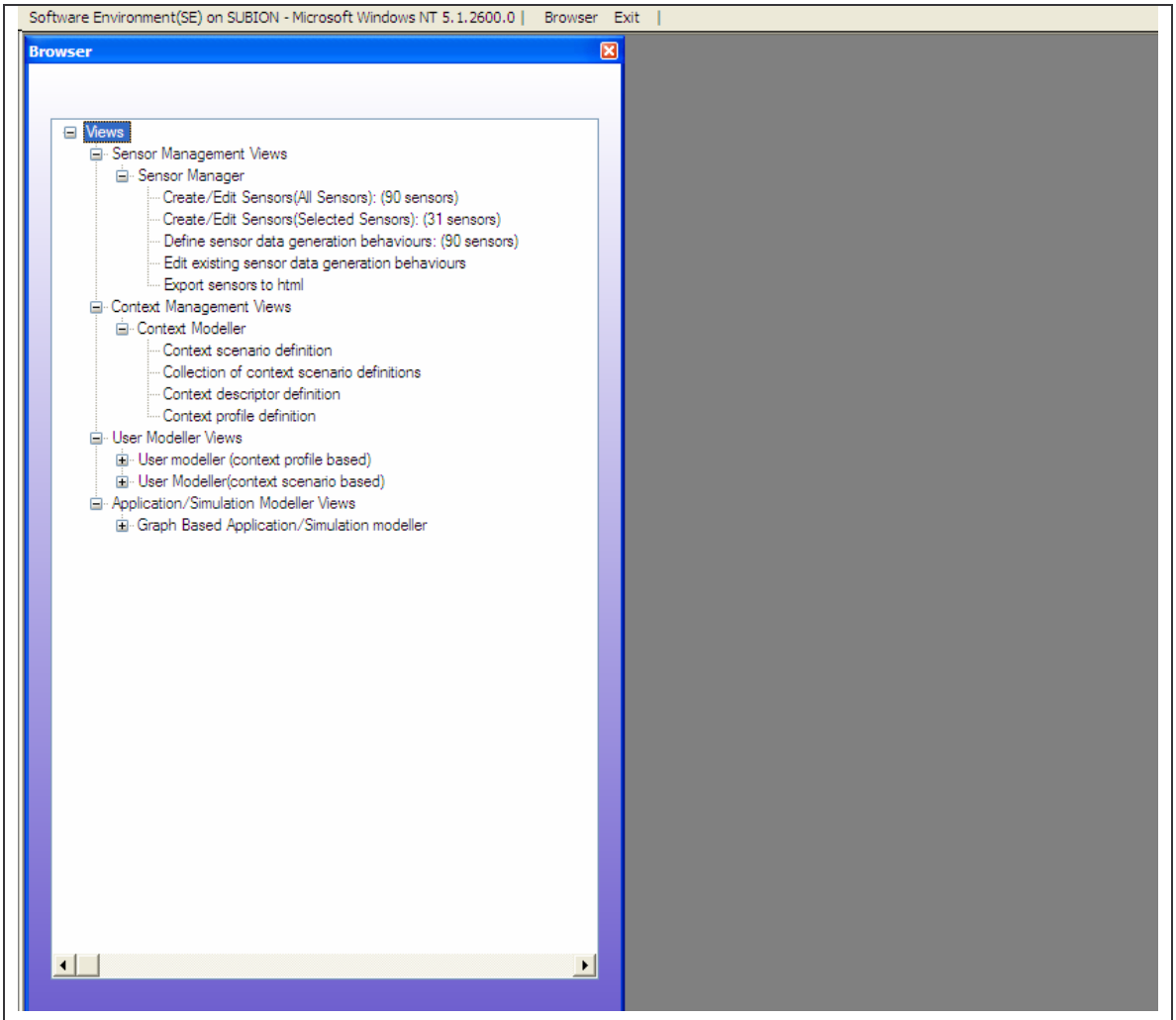


Figure 19. The software environment GUI view

Individuals can open and use multiple views that are needed for supporting their current task, which can be creating new sensors, updating the context model, building a new user model or construction and running of applications/simulations. As outlined in the software environment's specification, the software environment should be an environment which contains all the facilities needed for developing context aware applications. This implementation provides all the required facilities which are identified in section 5.1.3. For third generation based programming where writing code, compiling and debugging is necessary, existing tools such as Visual Studio [Microsoft, 2005b] are used.

5.3.3 The sensor management facility

The software environment's sensor management facility needs to fulfil the three requirements which are specified in section 5.1.3. This section describes how the current implementation meets these requirements.

Requirement 1: make it easy to integrate, test and use new sensor drivers as well as integrate existing ones

In order to make it simple to design, test and use sensor drivers, the software environment provides the concept of black boxes. Individuals implement sensor drivers as black boxes that produce data. Sensors are implemented as either data sources or data channels. Each component can be tested individually for both functionality and efficiency. Its functionality can be tested by checking its ability to produce an expected stream of data. Its efficiency can be tested by measuring the speed at which it performs its function and the amount of memory and CPU cycles it consumes.

Requirement 2: make it easy to use real sensor drivers in applications

The software environment provides implementations of sensor drivers in the form of either data sources (when a sensor driver captures data from a physical sensor) or data channels (when a sensor driver supports processing of recorded sensor data or accepts control commands in addition to capturing data from physical sensors). Examples are a video capture data channel, serial port data channel and audio capture data source. The video capture data channel represents a video camera from which video data can be captured. The serial port data channel acts as a source of raw data from any type of sensor that can be attached to a serial port. The audio capture data source acts as a source of audio data. Individuals can also use recorded raw data streams captured from existing sensors and re-use them in simulation or applications. The software environment provides data sources that enable this. Examples are a file data source and media player data source. The file data source enables the use of both text and binary raw sensor data streams. The media file data source enables the use of recorded video and audio streams while previewing them.

Using sensor drivers in applications or simulation is as easy as placing graph nodes which represent either data sources or data channels on a GUI canvas and connecting them to components that process sensor data as illustrated in section 5.3.6 and 5.4.

Requirement 3: provide the option of using virtual sensors when no real sensors are available

Individuals can use the GUI shown in Figure 20 to define new virtual sensors and use them in simulations as described in section 5.4. The GUI also enables the editing of existing sensors.

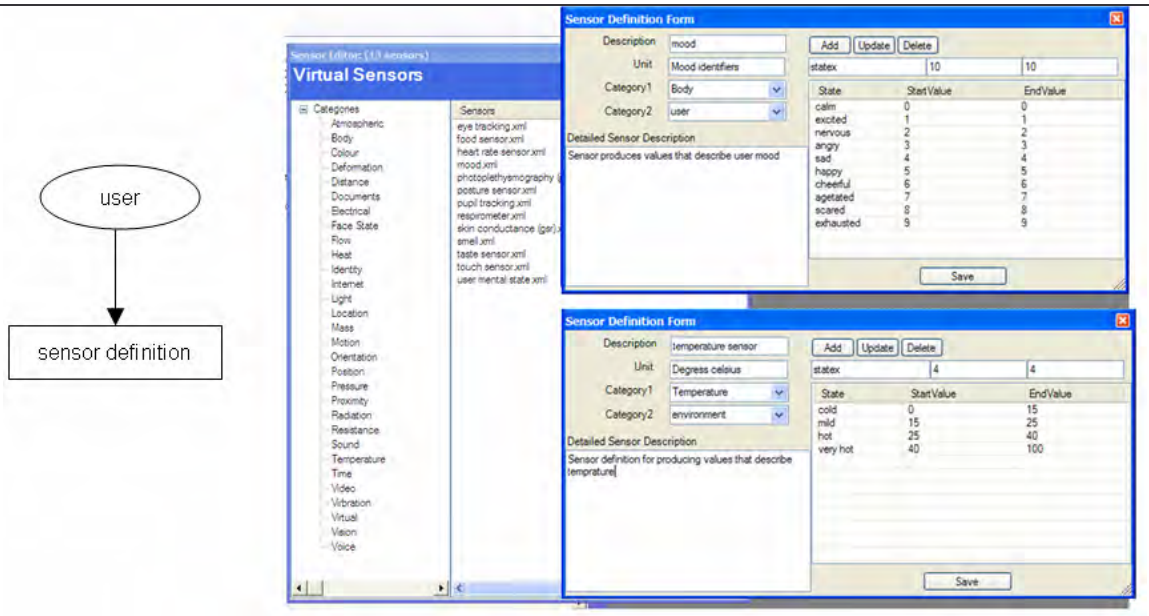


Figure 20. Virtual sensor editor

A virtual sensor is defined by specifying:

- a name and a detailed description for a sensor
- the unit used by a sensor
- two levels of categories which act as group IDs for the sensor. New sensor category identifiers can be added at senor definition or editing time
- known states of the sensor. These state labels are used to refer to individual values or intervals of raw sensor data values. For example, a temperature sensor data definition

will contain a state definition for *cold (0-15)*, *mild (15-25)*, *hot (25-40)* etc. whereas an object sensor definition will contain state definitions for *object1 (1)*, *object2 (2)*, *object3 (3)* etc.

Individuals can also synthesize streams of raw sensor data using virtual sensor definitions. The GUI shown in Figure 21 is used to create a script called a sensor data generation behaviour definition.

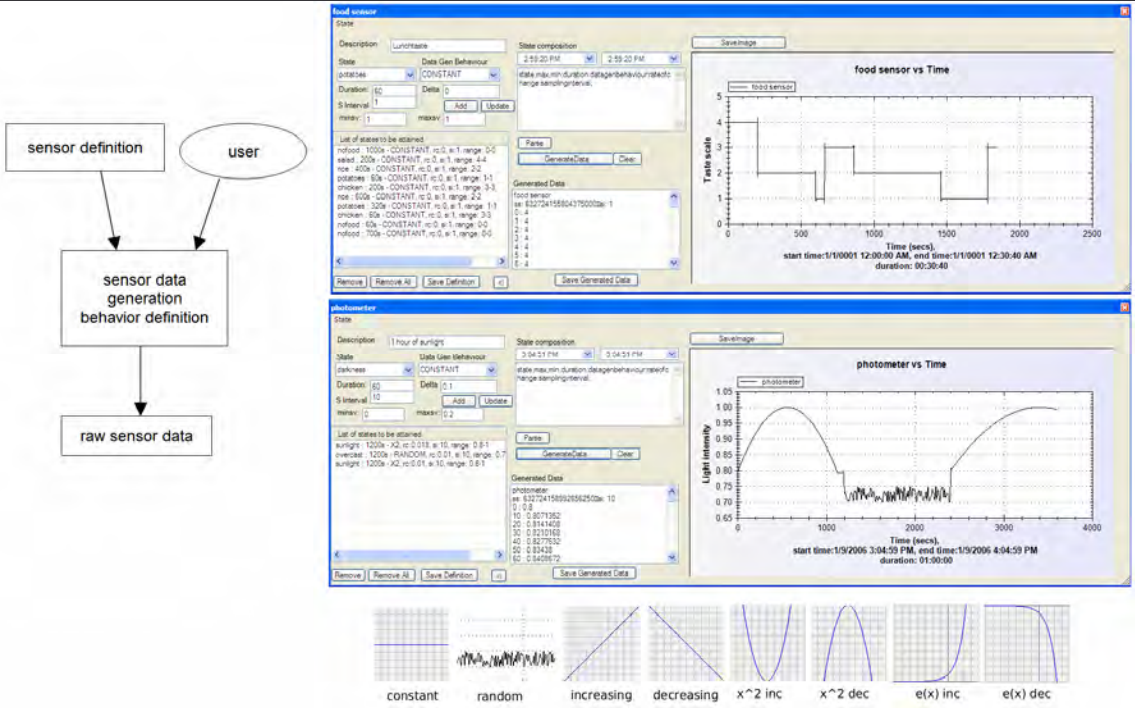


Figure 21. Sensor data generation behaviour definition GUI, generates data using piecewise functions

The script consists of a collection of piecewise mathematical functions which can be used to synthesize raw data streams from the state definitions contained in a virtual sensor definition. In the GUI shown in Figure 21, six functions (constant, random, increasing, decreasing, x^2, -x^2, exp(x) and -exp(x)) are available for composing sensor data streams. Once defined, sensor data generation behaviour definitions can be used in simulations.

5.3.4 The context management facility

The software environment's context management facility needs to fulfil three requirements which are specified in section 5.1.3. This section describes how the current implementation meets these requirements.

Requirement 1: make it easy to design, test and use new context interpreters as well as integrate existing ones

Context interpreter use in the software environment is similar to that of sensors as described in section 5.3.3.1. Individuals implement context interpreters as black boxes (data channels) which consume sensor data and produce context descriptors. To add a context interpreter to the system an individual only needs to implement a data channel which represents their particular context interpreter.

Requirement 2: make it easy to use context interpreters

The software environment provides implementations of a variety of context interpreters which can be used in constructing different applications or simulations. Implementations of context interpreters which are part of the software environment and the ease with which they can be used is demonstrated in section 5.4. Chapter 6 presents further implementations and use of context interpreters which are part of this work's machine based personal memory manager. Examples include a speech recognizer, a face recognizer and an audio analyzer.

Requirement 3: enable the building and use of context models

The context model used in this work consists of collections of context profiles. The software environment provides a method for individuals to define new context descriptors using the GUI shown in Figure 22.

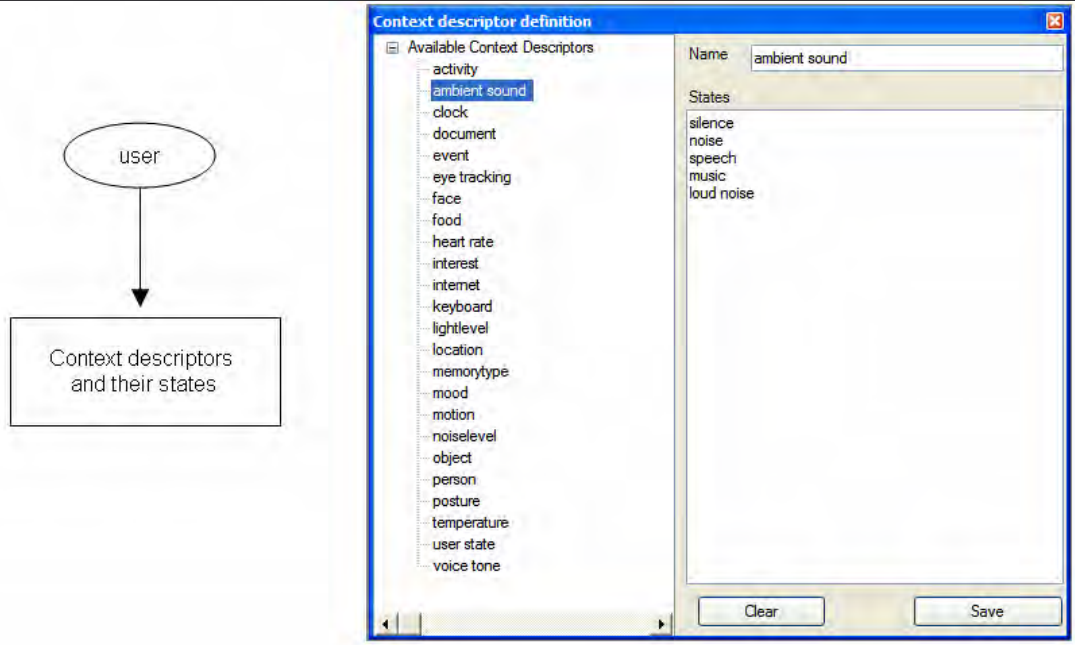


Figure 22. Context descriptor editor

The context management facility also enables individuals to create and edit context profiles using pre-defined context descriptors through the GUI shown in Figure 23. A context profile is created by grouping context descriptors which are known to represent a certain context profile and labelling them with a textual description.

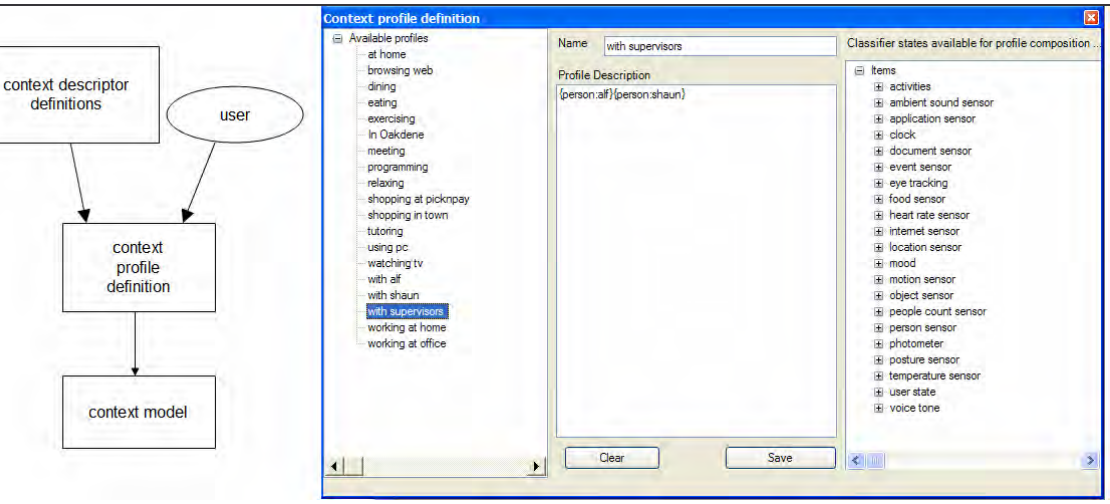


Figure 23. Context profile editor

An alternative method of modelling user context is in terms of raw sensor data rather than context descriptors. Sensor data generation behaviour definitions represent sources of raw sensor data. The GUI shown in Figure 24 enables the grouping of sensor data generation behaviour definitions to create a *context scenario definition* which is analogous to a context profile but represents collections of raw sensor data rather than context descriptors.

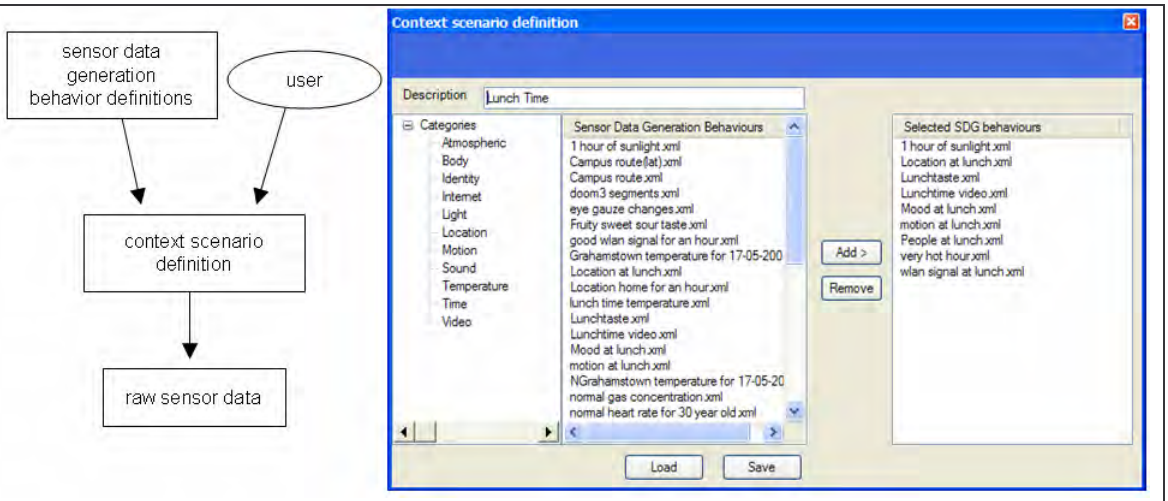


Figure 24. Context scenario editor

A visualization of the data produced by a context scenario definition is shown in Figure 25. The figure shows that such a definition represents multiple raw sensor data streams. The units of each graph are the same as those defined by the virtual sensor definition tool shown on Figure 21 (x axis – time, y axis - sensor unit). In the software environment, individuals can define any number of virtual sensors, create a sensor data generation behaviour, for each sensor, and save the collection of sensor data generation behaviours as a context scenario definition which can be used in a simulation.

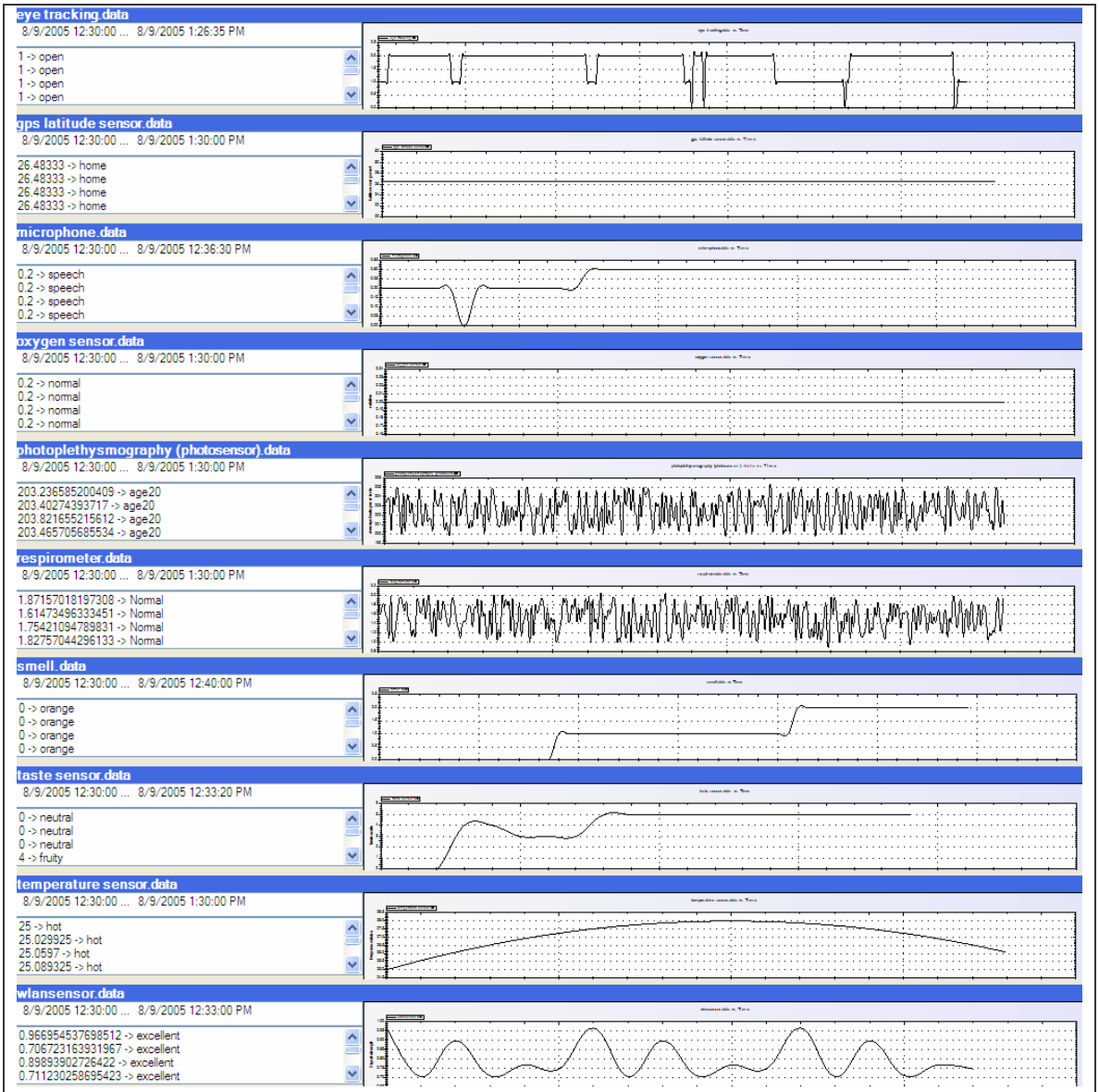


Figure 25. Visualization of data produced by a context scenario definition using multiple virtual sensors

5.3.5 The user modelling facility

The software environment's user modelling facility enables modelling the state of a user and his environment over periods of time using the context model built with the context management facility. The GUI shown in Figure 26 is used for this purpose. A user model is built by constructing a graph. Each node of the graph represents the states of a user or his environment at a particular moment in time defined as one or more context profiles. The vertical position of a node on the graph (its y coordinate) represents the time of day

(0:00 to 23:59h). The vertical distance between two nodes A and B (where the y coordinate of A < B) represents the time period during which the context profiles that are represented by node A are active.

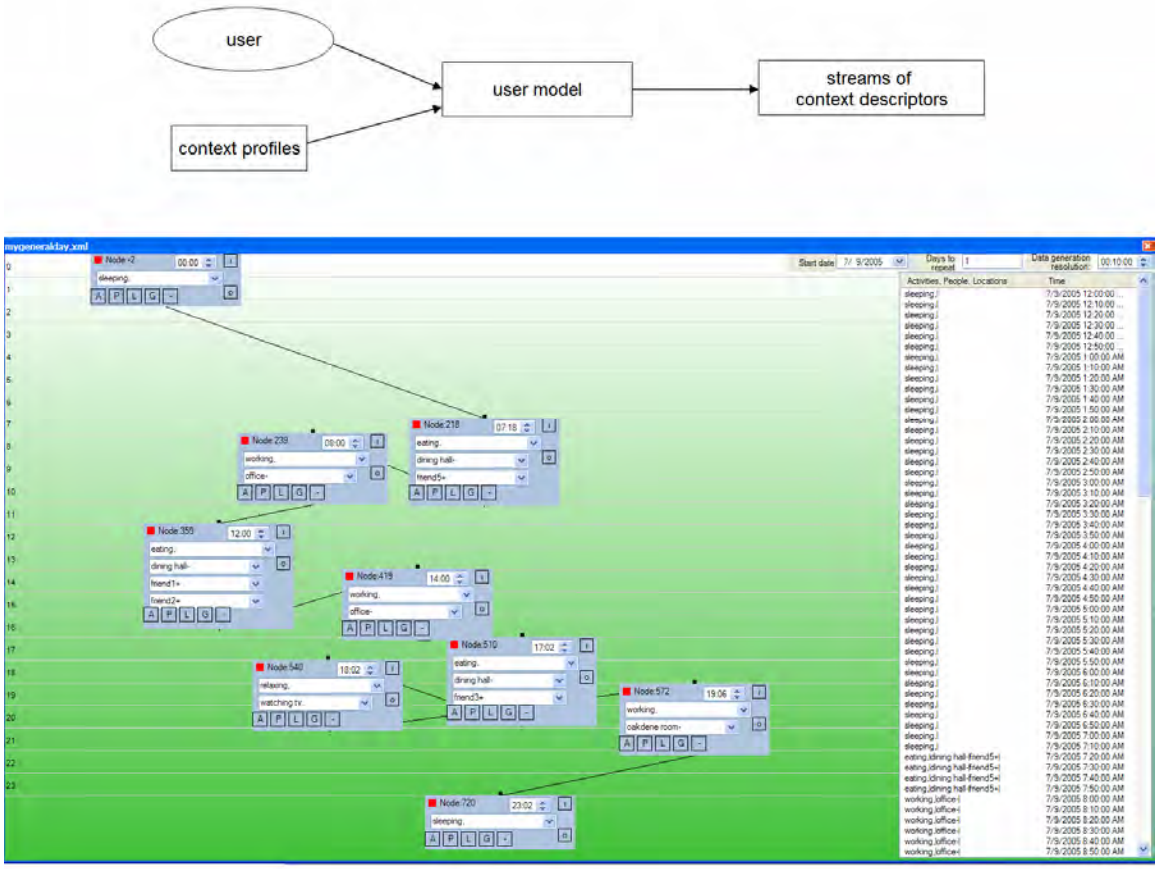


Figure 26. The software environment's user modelling facility

Once a user model is built, it can be used to generate streams of context descriptor states. This is done by mapping the context profiles which are associated with each graph node to context descriptor states using a context model. The generated streams of context descriptor states can, for example, be used to tag an individual's memory fragments which are formed during the time period that the user model represents. This method of memory tagging is used in the manual context descriptor extraction process described in chapter 6. This facility meets the requirement for a user modelling facility set in section 5.1.3 since it makes it easy to model a user's states using an existing context model.

5.3.6 The application and simulation modelling facility

The software environment provides a data flow based application and simulation modelling facility which, through a GUI:

- makes it easy to run simulations and construct applications
- enables simulations involving context aware applications and user models to be performed
- enables rapid application development through visual programming and component re-use

This meets the requirements of the software environment specified in section 5.1.3. Proof that the application and simulation modelling facility provides the claimed functionality is provided in section 5.4 and chapter 6. Data flow based modelling is chosen as the main modelling approach in the software environment because it enables dynamic systems to be modelled interactively.

The software environment provides a graph and data flow based modeller called the graph manager which is shown in Figure 27. Individuals create simulations/applications by constructing graphs using data sources, data sinks and data channels. Prior to individuals composing graphs, the graph manager interacts with the software environment's resource manager and dynamically loads existing data sinks, data sources and data channels. These components represent graph nodes. Each black box can render a visual representation of itself. If it does not, a default rectangular representation is provided by the component's base class.

In the graph manager, individuals compose graph models by placing graph nodes on the graph manager's canvas and connecting them up. When a node is placed on the graph canvas, the graph manager calls its *Load()* method, when a node is removed from the graph canvas the graph manager calls its *UnLoad()* method. These methods enable a component to perform initialization and clean-up operations.

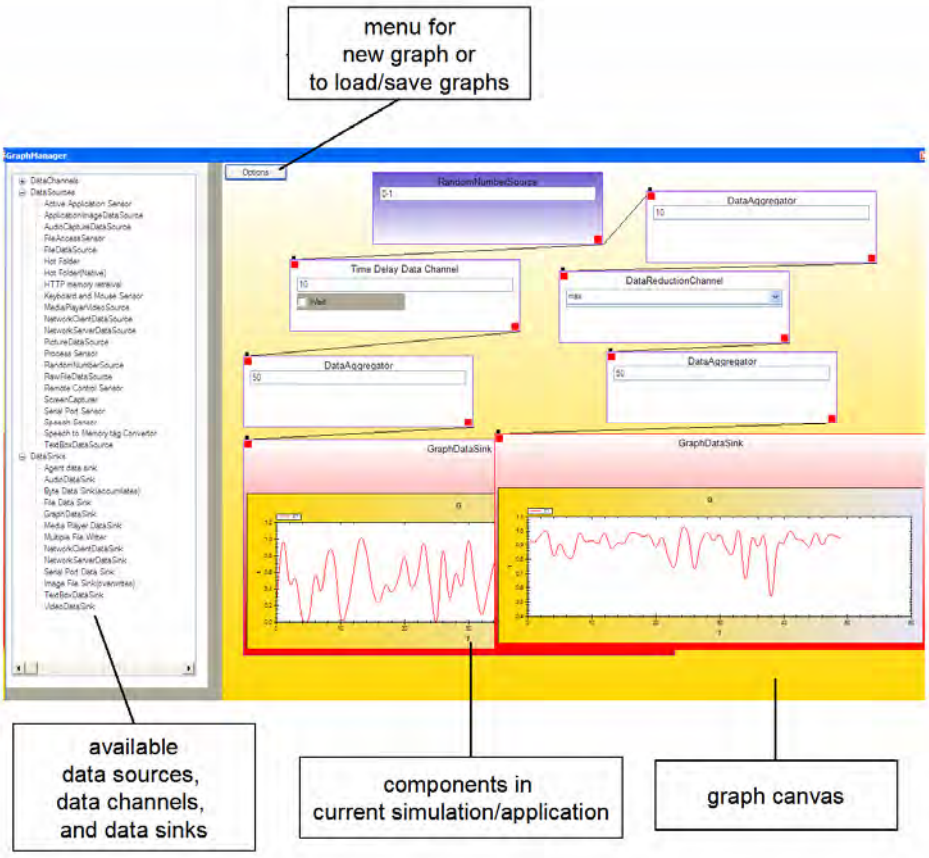


Figure 27. The graph manager (a graph and data flow based application and simulation modeller)

When an individual creates a connection between two nodes, the graph manager calls the *GetSample()* method of the component associated to the node that is the source of a connection and passes the retrieved sample to the *CanConsume()* method of the component associated to the node which is the sink node. If the *CanConsume()* method returns true, a connection can be created, otherwise the user is notified that the two graph nodes are incompatible and the connection creation aborted. Unlike existing data flow systems which are identified in chapter 3 and section 5.2.1.3, individuals who implement graph nodes or those that model systems using them, do not need to be concerned with the number of connection points (ports) that exist between two nodes. Connection and data routing between nodes is handled automatically and transparently by the graph manager. Queues are used to buffer data from multiple data sources which are to be processed by a data channel or data sink.

Once a graph is constructed, it can be saved as a graph model. A graph mode consists of a list of graph nodes and their coordinates on the canvas as well as a list of connections between nodes. The saved model can be reloaded into the graph manager at a later time to re-run applications or simulations.

5.4 Simulation and application modelling in the software environment

Currently, the software environment contains 15 data sources, 12 data sinks and 50 data channels which represent components such as sensor drivers, context interpreters and memory presentation interfaces. Using these components, users can interactively compose graphs that represent simulations or applications. Examples of the kinds of systems that can be composed and modelled in the software environment are presented in this section. They serve as proof that the software environment enables visual composition of systems rapidly while enabling component re-use. Further models which are implemented as part of a machine-based memory manager are presented in chapter 6. When considering the examples presented in this section, the following points should be noted.

- A system is represented by a graph model which consists of data sources, data sinks or data channels.
- The black boxes which are used in the examples are created independently of the larger system that they are part of.
- Individually, components have little value but when connected together in groups as illustrated in this section, they can perform useful functions.
- When composing new systems using the graph manager, pre-built binary components are used, therefore no third generation based code writing and compilation is necessary.
- The black boxes which are used in each system can be removed or replaced while other parts of a system are still running.

5.4.1 Context descriptor extraction

This section presents example graph models that demonstrate context descriptor extraction. Context descriptors are an essential part of the model for machine-based memory management which is presented in chapter 4. The context descriptor extraction process is made possible by sensors and context interpreters as described by the model in chapter 4.

5.4.1.1 Sensor data processing and visualization prior to context descriptor extraction

A graph model which demonstrates raw sensor data processing and visualization is shown in Figure 28. Sensor data pre-processing is essential to a context interpreter when it extracts context descriptors. Visualization is important for an individual who is developing a context interpreter. The kind of sensor data pre-processing which is demonstrated is similar to Schimdt's cue concept [Schmidt, 2002] that is considered in chapter 3. The graph model contains the following components:

- a fictitious sensor data source – a random number generator
- a time delay data channel – controls the rate at which sensor data is sampled
- 3 data aggregator data channels – collect a specified number of data samples
- 1 data reduction channel – averages data samples
- 2 data visualization data sinks - which use ZedGraph [Champion, 2004] to visualise the sensor data stream

Figure 28. Sensor data processing and visualization

5.4.1.2 Context interpreter simulation

The software environment contains a facility which enables the definition and use of virtual sensors. Section 5.3.3 has presented the implementation of this facility. Figure 29 shows how a simulation can be run by combining virtual sensor definitions and real sensor data. The example makes use of data recorded from a real sensor and a number of sensor data generation behaviours which are defined using the sensor management facility of the software environment. The graph model consists of:

- a random number data source and time delay data channel - together act as a clock which controls the rate at which the simulation runs
- a multiple sensor data channel - produces sensor data samples as defined by sensor data generation behaviours (based on virtual sensor definitions)
- a raw sensor data source channel - a source of recorded sensor data. Data is read from a text file which contains recorded GPS data

- a GPS data formatter channel – extracts latitude and longitude data from GPS strings
- a context interpreter – a simple context interpreter which fuses multiple raw sensor data streams to produce new a context descriptor. In the example, it outputs the total number of raw sensor data streams which have changed their state in the last second.
- a text box data sink – displays the output of the context descriptor

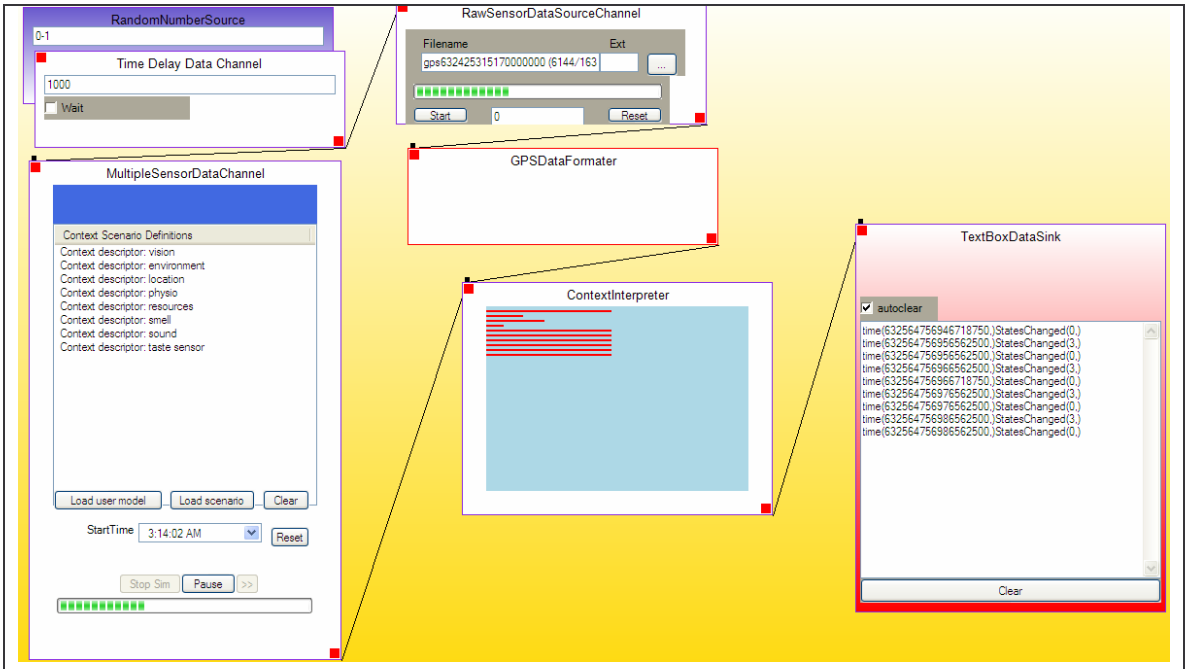


Figure 29. Context interpretation using virtual and real sensor data

5.4.1.3 Simultaneous extraction of context descriptors from multiple sensors

In real world applications context descriptors need to be extracted in real time from a variety of sensors. Figure 30 shows an example graph model that demonstrates context descriptor extraction from multiple sensors. The graph makes use of the following components:

- a text box data source – a source of text data from an input device such as a keyboard
- a picture data source – a source of visual sensor data. In the example, the source is a picture taken with a cellular phone camera
- an OCR data channel – a context interpreter which recognizes text from video, implemented using the Microsoft office OCR library [MODI, 2005]

- a speech sensor – is both a sensor (microphone) and a context interpreter which converts speech to text, implemented using the Microsoft speech SDK [Microsoft Speech, 2005]. The speech recognizer uses a limited grammar, which increases its accuracy, which in this case only contains the phrase, "hellow world"
- a text data sink – displays recognized text

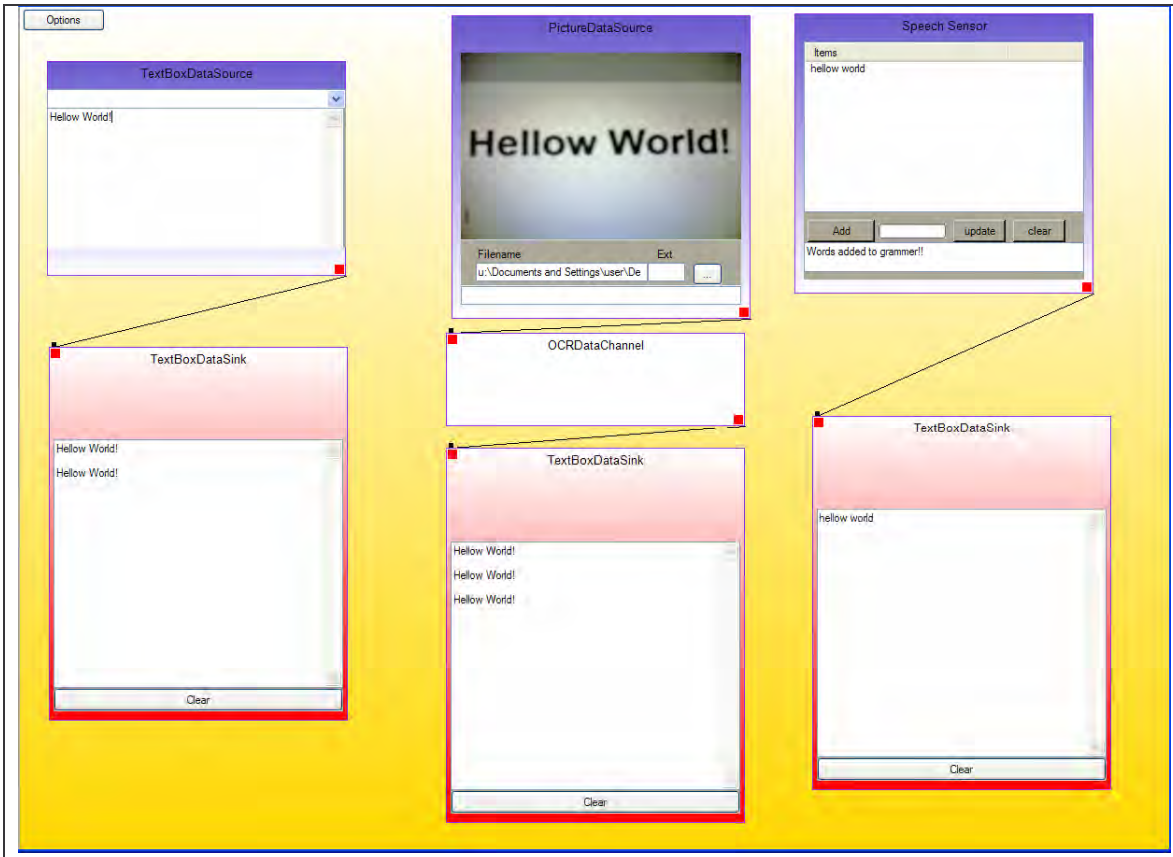


Figure 30. Context descriptor extraction from multiple sensors

The example shows how a message such as "hellow world" can be extracted using three different sensor modalities. Each sensor modality has its advantages. Speech is natural. Keyboard requires some user attention. An OCR does not require mechanical user input. Humans can easily interpret what the message, "hellow world" means. If the message is changed from "hellow world" to "<greeting> hellow world" a machine-based context interpreter that recognizes semantic tags such as "<greeting>" can interpret the semantic meaning of the message as a greeting.

5.4.1.4 Extracting context descriptors from a single sensor data source

Sensor data such as video contains a lot of information about the environment. Figure 31 shows an example graph model which performs multiple types of sensor data pre-processing and context descriptor extraction all at the same time, using a single stream of video data.

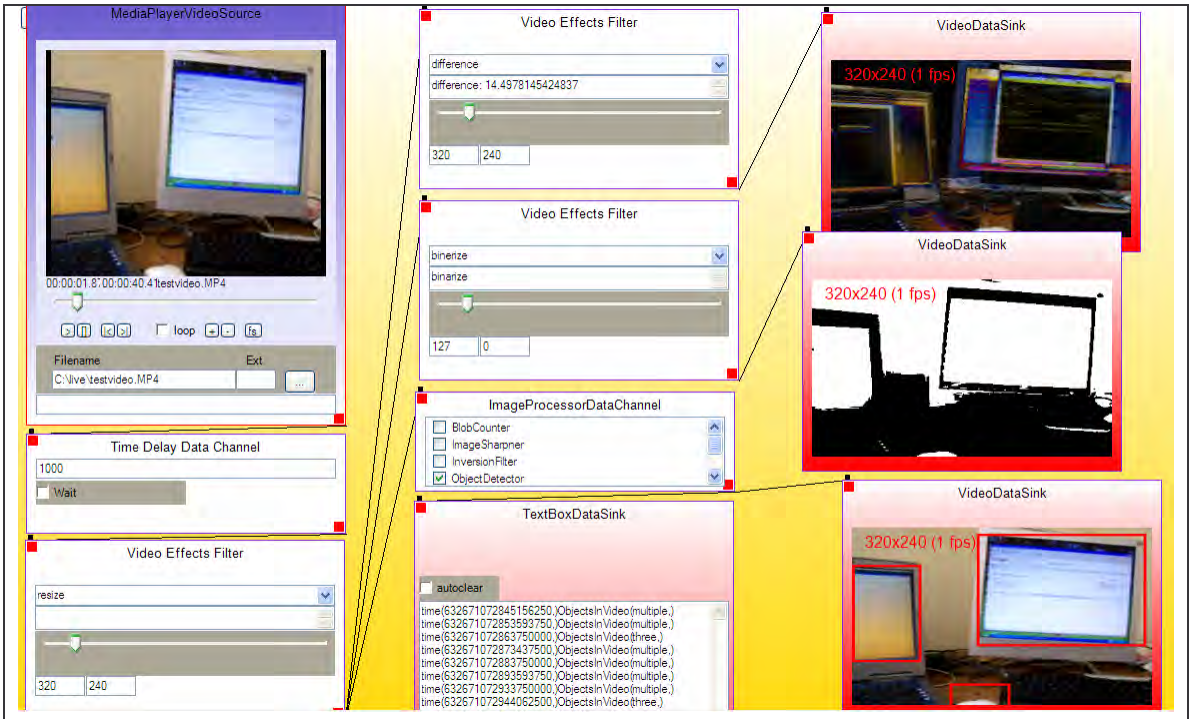


Figure 31. Video processing

The graph model consists of:

- a media player video source – a source of recorded video data
- a time delay data channel – controls the rate at which video data sensor is sampled
- a video effects filter – configured to resize images to 320x240 pixels to speed up video processing
- a second video effects filter – configured to apply a binarization operation to images. It converts colour images to monochrome. A binarization filter is usually applied to frames as a precursor to performing further video analysis. For example, it is used by

Rekimoto [Rekimoto *et al,* 2000] to identify his Cybercode visual tags which act as context descriptors

- an image processor data channel - analyzes images to identify the number of objects contained in video. It outputs one, two, three or multiple as object count context descriptors

5.4.2 Interactive applications

The software environment can be used to compose interactive applications. This section presents three example applications: a video memory capture application, an email notification agent and a blackboard application. All systems are composed using a combination of data sources, data channels and data sinks which act as building blocks for a bigger system. This examples show that applications can quickly be composed in the software environment.

5.4.2.1 A video memory capture application

A memory capture application which makes use of a circular buffer to store the last *n* video frames that are seen by an individual is shown in Figure 32. The idea is that when an individual sees something interesting happening he tells his system to save what he just saw and the system saves the visual memories which are currently in the circular buffer. In this way, a person can capture interesting visual memories as he goes about his daily life. The advantage of the system is that the individual only captures what he wants to capture. The graph model makes use of the following components:

- 2 video capture data channels – sources of video data from two separate cameras
- a video combiner data channel – combines video data from the two cameras
- a speech sensor – accepts commands from a user such as “computer capture”
- a video data sink – a preview panel for captured video frames

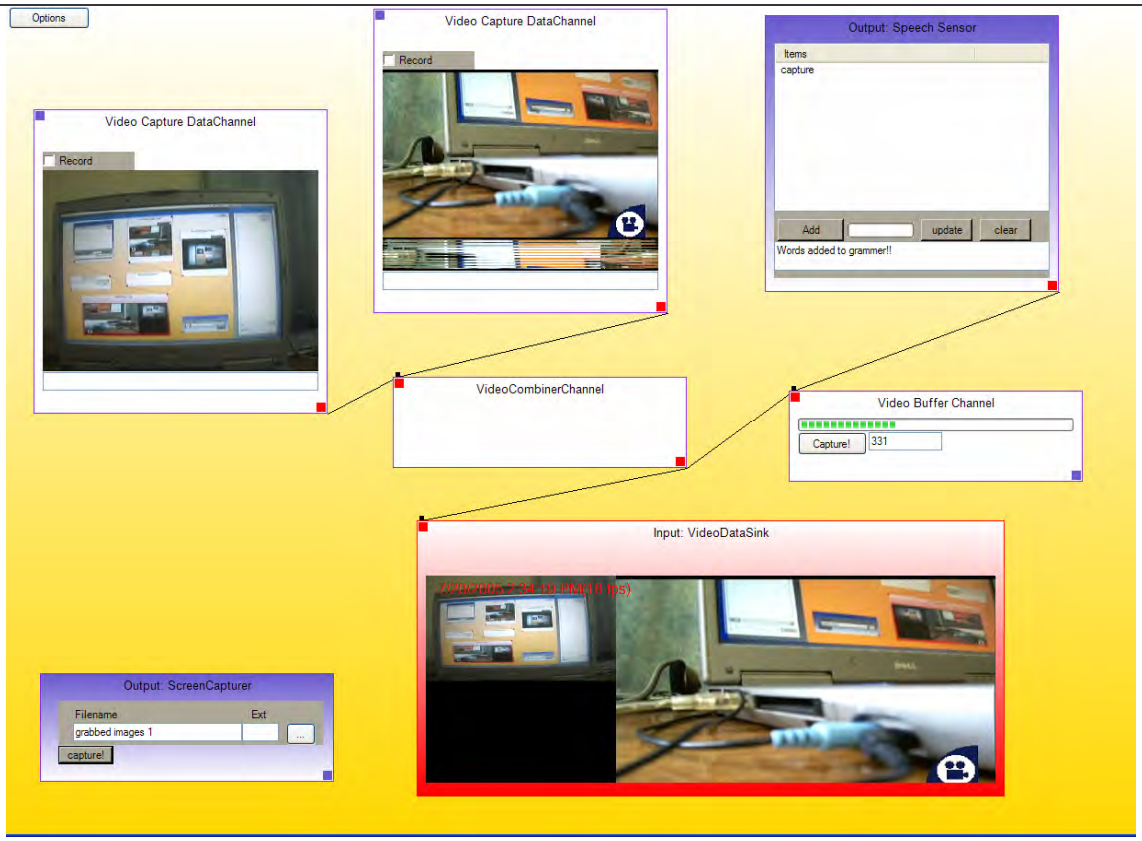


Figure 32. Selective video memory capture

5.4.2.2 Email notification agent and time announcer

A second interactive application is shown in Figure 33. The graph model represents an email notification agent as well as a time announcer. The graph model consists of the following components:

- a speech sensor and a text box data source – accepts commands from users
- a command interpreter data channel – interprets commands; either a request for the current time or a request for messages in an inbox
- an email sensor data channel – checks messages in a remote mailbox. Also sends notifications when new mail arrives
- an agent data sink – contains an avatar [Rao, 2000] which announces notification messages to users

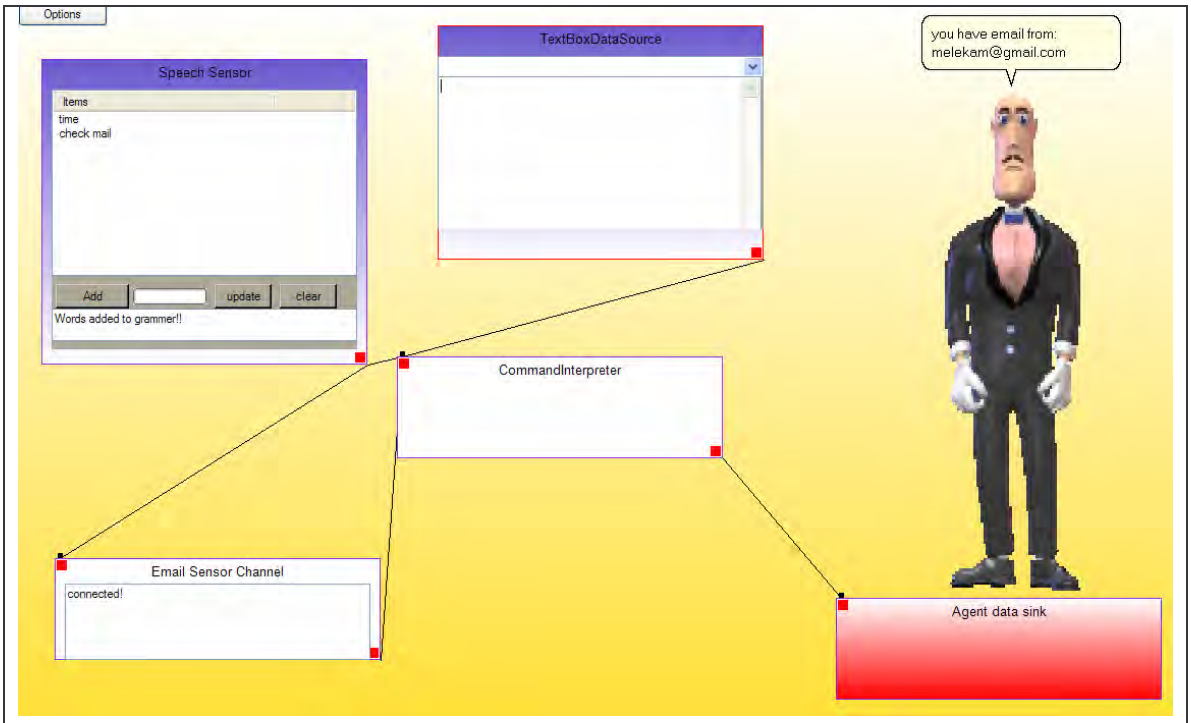


Figure 33. Email notifier and time announcer

The agent's avatar is also capable of performing a number of actions which can be used to show emotion such as clapping, walking, moving and making noises. Giving machines emotions can make them more friendly and useful to people [Bartneck, 2001] [Sebe *et al*, 2005].

5.4.2.3 A blackboard application

Blackboard applications are used in collaborative work environments such as those Winograd [Winograd, 2001] discusses. Individuals in a room, or those spread over remote locations, can each use a blackboard tablet to hold discussions and exchange ideas. Whenever one individual writes on a tablet the changes are displayed on everyone's tablet. An individual can also save the current contents of a tablet by clicking on a button. Figure 34 shows a graph model that represents such an application.

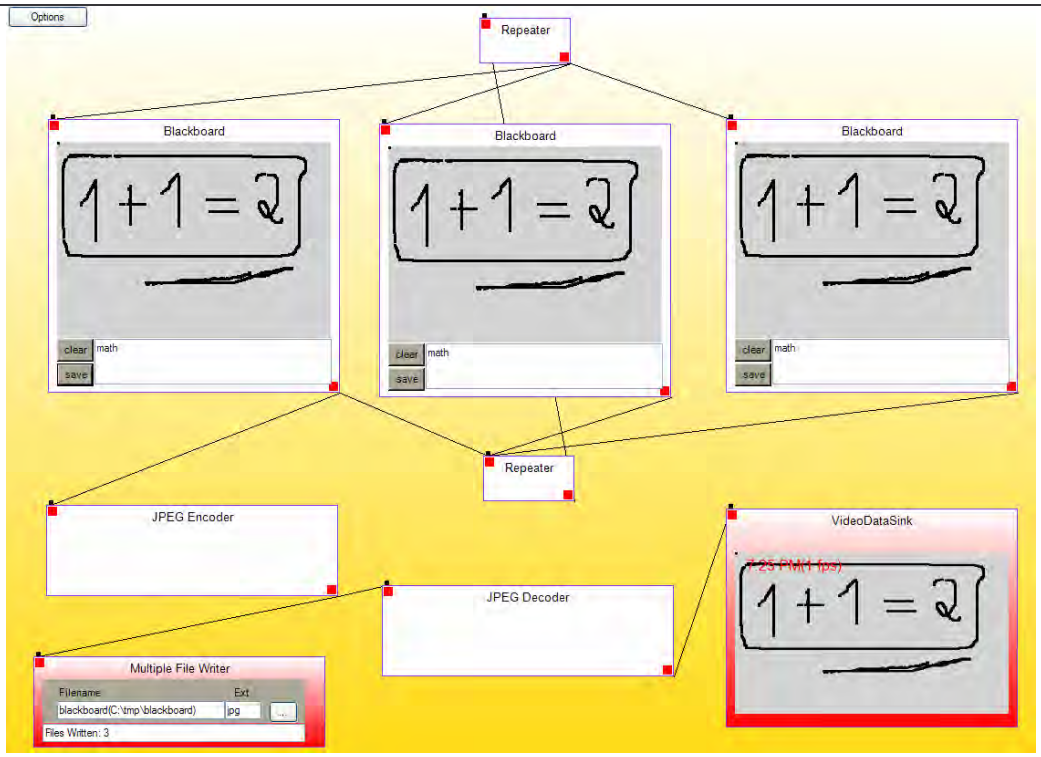


Figure 34. Blackboard example

The graph model consists of the following components:

- a black board data channel – represents the black board on which a user writes
- a repeater data channel – repeats any data that it receives. It represents a component which is used to make the model clearer visually but does not contribute to the systems functionality
- a JPEG encoder data channel – encodes data it receives using JPEG compression
- a JPEG decoder data channel – decodes JPEG data
- a multiple file writer – saves data it receives as individual time stamped files
- a video data sink – previews the currently saved image from blackboard one

5.4.3 Distributing applications and simulations over a network

The graph modeller also enables modelling of systems which are distributed over multiple machines. It also enables systems to transparently use a networked architecture.

Figure 35 shows a graph model that demonstrates the multiplexing of video and text data over a network connection using data channels that represent network end points.

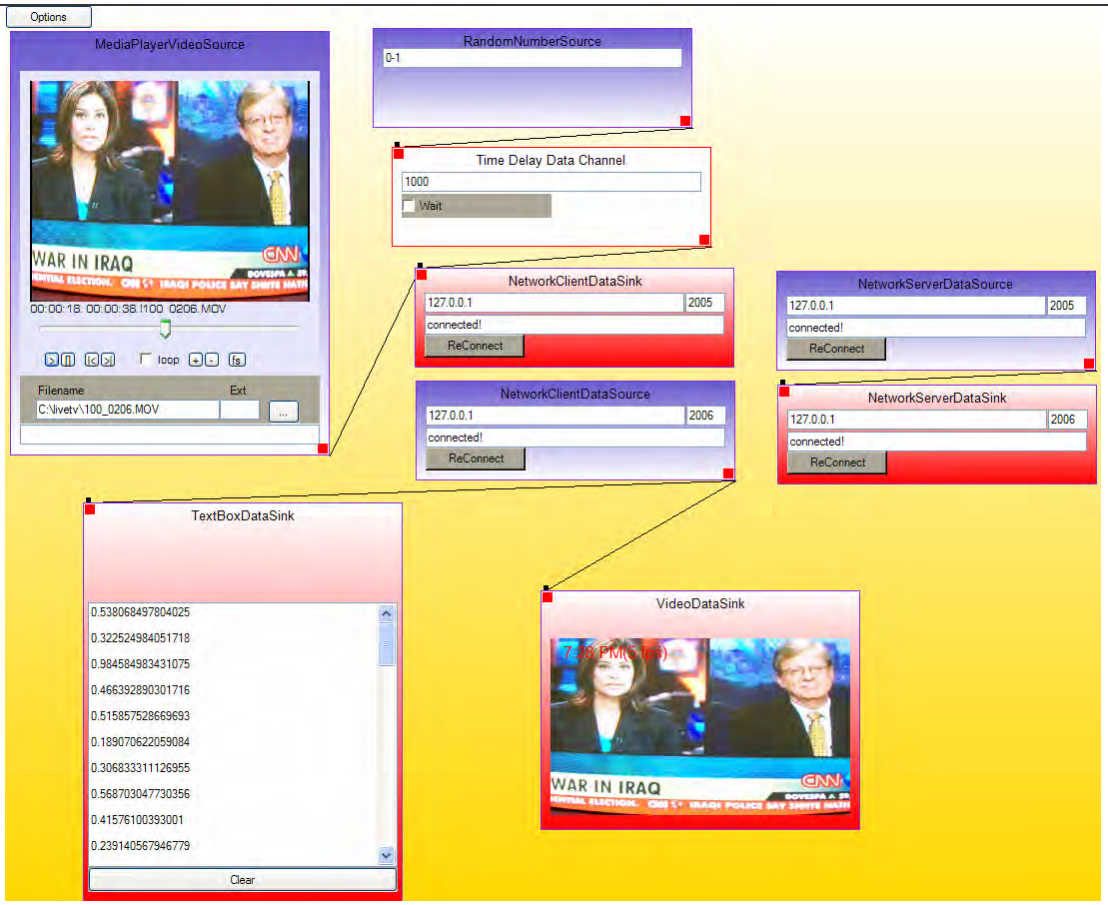


Figure 35. Running graphs over networks

The graph model consists of the following components:

- a media player video source – a source of video data
- a random number data source and time delay data channel – a source of text data
- network client data sink – an exit out to the remote network
- network server data source –an entry into the remote network
- network server data sink – an exit back to the local network
- network client data source – an entry back into the local host

- text box data sink – display for text data which is received on the remote end of a network
- a video data sink – display for video data which is received on the remote end of a network

5.4.4 Running multiple graphs at the same time

A number of individual graph examples have been presented thus far. Each example is composed and run separately. A high resolution screen is recommended when composing systems. During development a display with a resolution of 1920x1600 was used and provided a good environment for modelling. The graph manager has also been used on systems with lower resolutions such as 1280x768 which provide a smaller modelling space.

To compose, as well as run, a more complex system, the graph manager includes an application manager that facilitates the running of multiple graph models at the same time. Graph models can communicate with each other using network components or a shared database. Figure 36 shows all the example graphs presented in this section being run at the same time. Selecting an item opens that particular graph model.

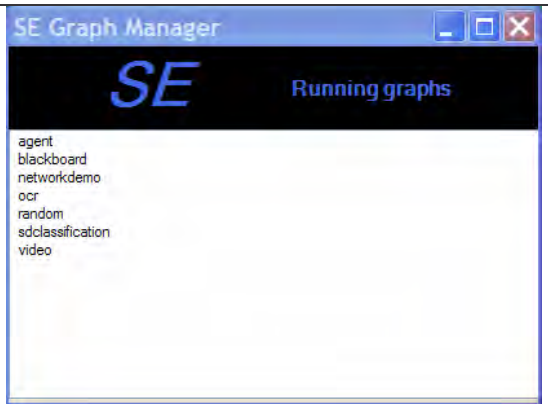


Figure 36. Multiple graph model running application

5.4.5 Graph modeller performance

All of the example applications presented in the preceding sections run in real time. Examples that involve video processing use 50% or more of CPU resources and memory usage does not exceed 80MB on a laptop containing 512 MB of RAM and running at 1.1 GHz. The data output rate of a component (data source or data channel) is dependent on its efficiency. For example, for data channels that process video data and output video data, the output frame rate of the processed video is a measure of the efficiency of the data channel. Chapter 7 presents an evaluation that tests the performance of various graph models which are part of an experimental machine-based personal memory manager.

6 Implementation of an experimental machine-based memory manager

The previous chapter presented the software environment, a system building tool which provides the facilities needed to develop context aware systems. This chapter describes the implementation of an experimental machine-based memory manager, which is based on the model presented in chapter 4, using the software environment. The machine-based memory manager is built using several black boxes which are connected to form larger subsystems as described in chapter 5. A visual programming tool called a graph manager (described in chapter 5) is used to build the various subsystems of a machine-based memory manager using these black boxes. This approach to system building greatly simplified the building of the machine-based memory manager which represents a complex system with several subsystems.

In this chapter, the composition of the machine-based memory manager is specified in terms of instances of the entities which are part of the model presented in chapter 4. The machine based memory manager's hardware platform is described. The implementation of its software components is presented.

6.1 The composition of the machine-based memory manager

According to the model which is presented in chapter 4, a machine-based memory manager consists of a memory manager, memory producers and memory consumers as well as memory fragments and context descriptors which these entities either produce or consume. This section describes instances of these entities which form part of this implementation of a machine-based memory manager.

6.1.1 The memory manager

According to the model, the memory manager is an entity which stores and facilitates the retrieval of memory fragments with the help of a context aware file system (CAFS). The

CAFS consists of a context descriptor layer and a quanta file system layer. Section 6.3 describes the implementation of the CAFS and how the memory fragment storage and retrieval processes are facilitated using the CAFS.

6.1.2 Memory producers for context descriptor extraction

A variety of context descriptors are used by the machine-based memory manager. This section presents instances of memory producers that identify context descriptors from sensor data, either automatically or manually.

6.1.2.1 Automatic context descriptor producers

Memory producers (sensors and context interpreters) which are used for automatic context descriptor extraction by the experimental machine-based memory manager as well as the context descriptors they produce are shown in Figure 37.

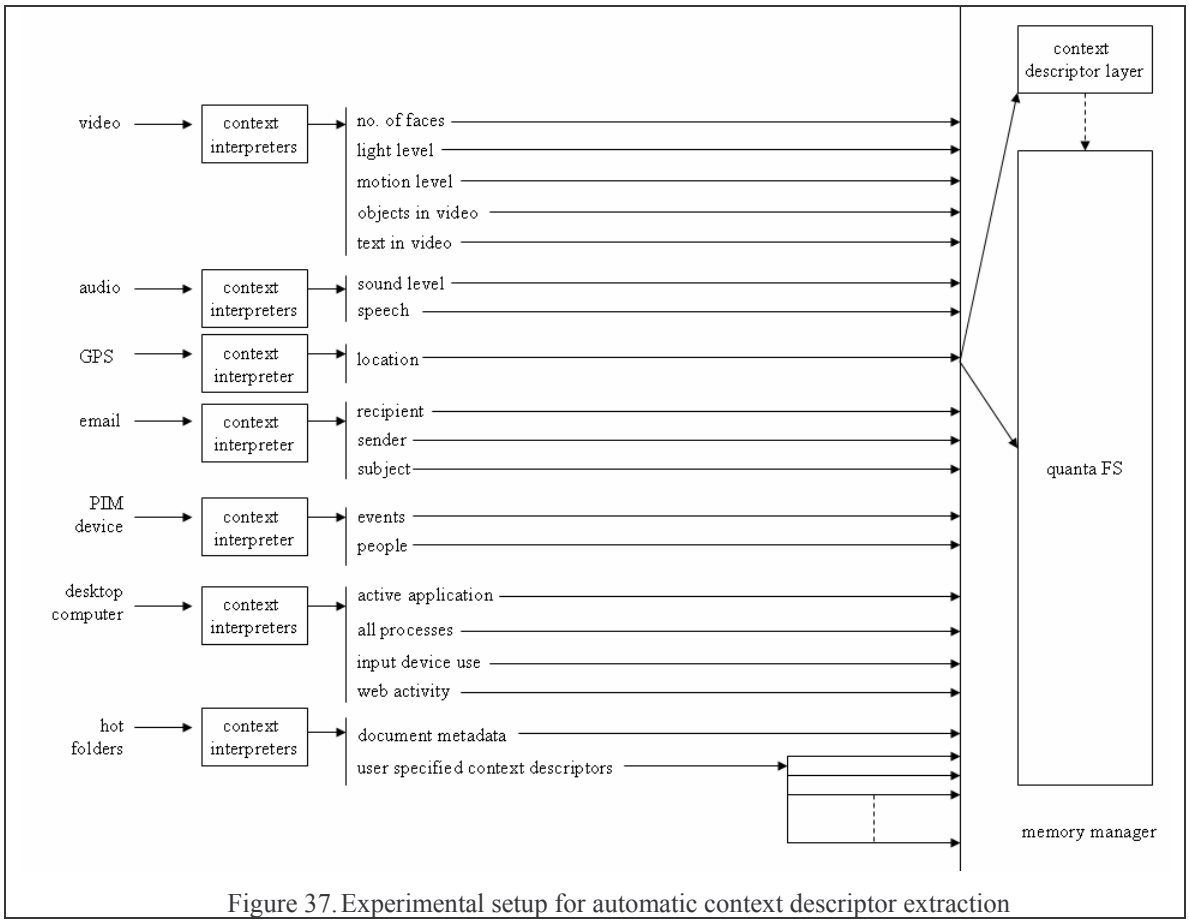


Figure 37. Experimental setup for automatic context descriptor extraction

A number of memory producers are used by the machine-based memory manager. Each memory producer extracts one or more context descriptors from sensor data and passes them to the memory manager for storage. Context descriptors are stored in both the context layer and quanta file system layer of the context aware file system. While each memory producer can operate autonomously, hot folders require some attention from a user. A user needs to place captured sensor data into a hot folder, which will then be absorbed automatically and analyzed. Functionally, hot folders resemble labelled file drawers where the label represents one or more context descriptors. Unlike drawers in an office, there is no limitation on the number of hot folders that can be used.

6.1.2.2 Manual context descriptor producers

Not all context descriptors can be extracted automatically by the experimental setup shown in Figure 37. Figure 38 shows two memory producers that identify context descriptors from commands given by users through speech or a remote control. Users do not supply unstructured text annotation, they provide context profile or context descriptor descriptions which in future can be substituted with context descriptors from automatic context interpreters. The context profile interpreter maps context profiles to context descriptors using a context model. It also contains a context descriptor stream generator which continually generates context descriptors that are mapped from the current, manually specified context profile.

6.1.3 Memory producers for memory fragment capture

In addition to identifying context descriptors, the machine-based memory manager also captures memory fragments. Figure 39 shows the various types of memory fragments which are captured by the experimental system. Each memory producer captures memory fragments and passes them to the memory manager for storage. Memory fragments are stored in the quanta file system layer of the context aware file system. Hot folders provide a means of capturing memory fragments, other than those captured by the sensors which are part of the machine-based memory manager, from external sources.

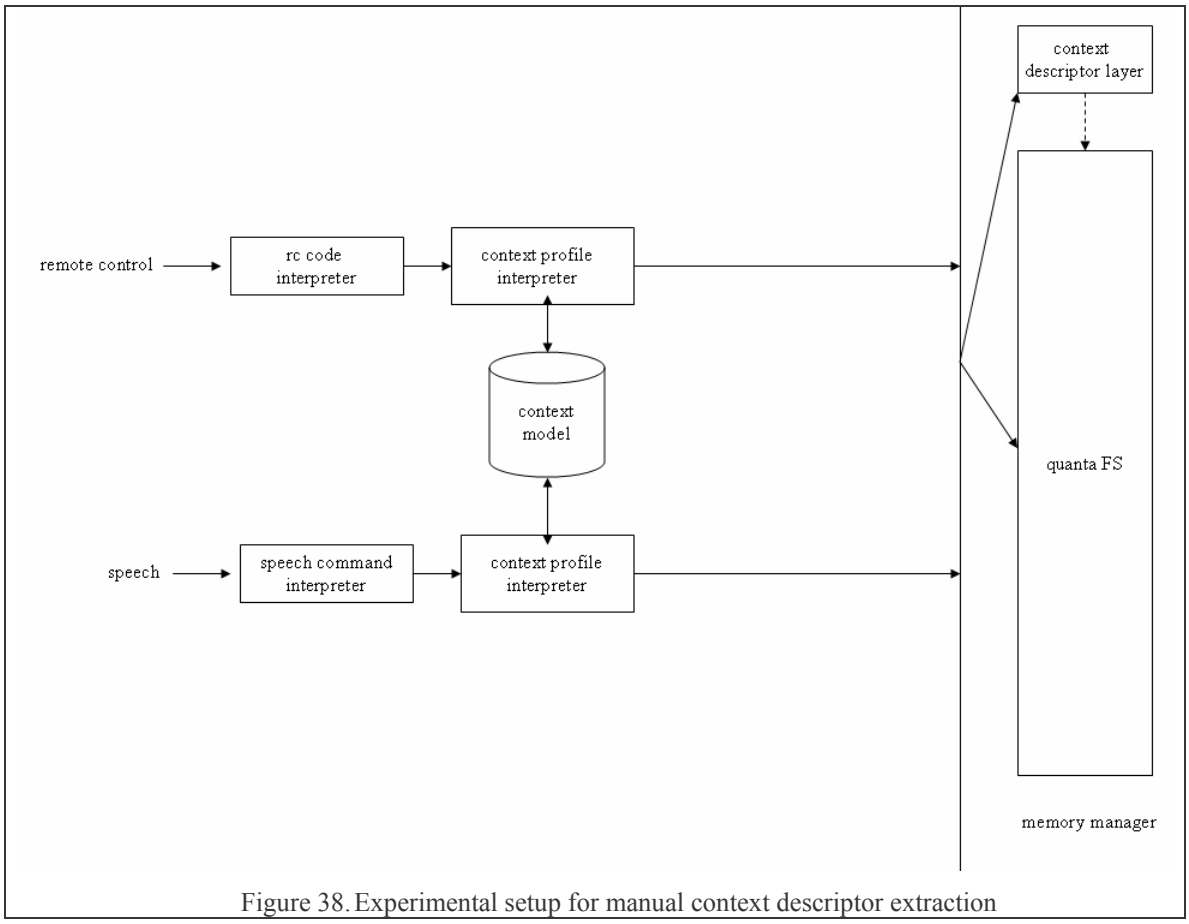


Figure 38. Experimental setup for manual context descriptor extraction

6.1.4 Memory consumers for memory fragment retrieval and consumption

The memory manager needs to enable users to retrieve and consume their memories in a number of ways. An ideal machine-based memory manager uses a brain computer interface (BCI) and a virtual display (see through eye mounted display described in section 4.5.4) for memory retrieval and consumption. A BCI is ideal because it connects a machine to an individual's brain so that it can be controlled by thought. A virtual display is ideal because it enables individuals to consume visual memories privately. However since these interfaces are not currently available in convenient forms they are not used in this implementation. In their place, the memory consumers shown in Figure 40 are used for memory retrieval and consumption.

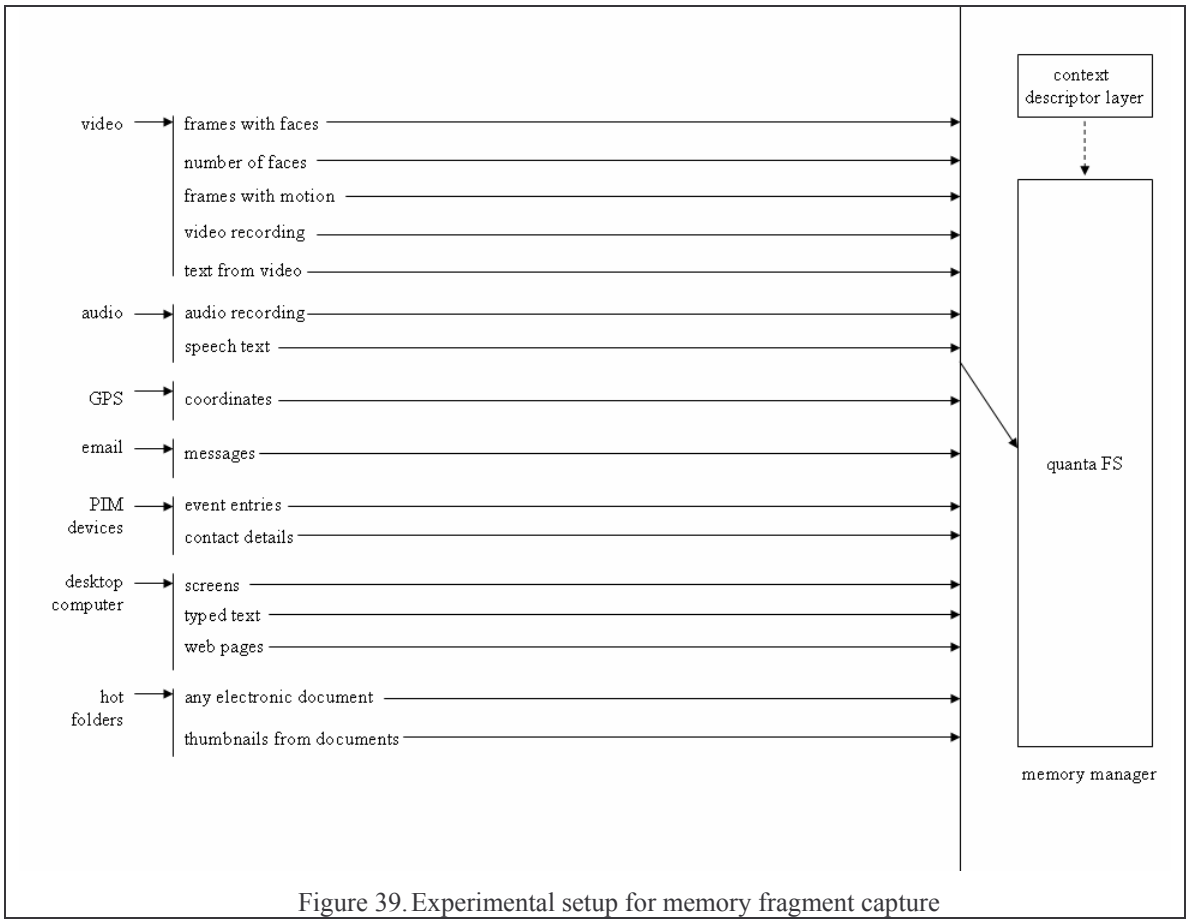


Figure 39. Experimental setup for memory fragment capture

- *Memory retrieval interfaces*

A user supplies context descriptors which are to be used for memory retrieval by the machine-based memory manager through the interfaces shown in Figure 40. They consist of: (1) a speech or remote control which is described in manual context descriptor extraction in section 6.1.2.2, (2) a PDA which enables memory retrieval everywhere because it is small and portable, (3) a keyboard which is used on desktop computers for manual context descriptor extraction and (4) hot folders which are described in automatic context descriptor extraction in section 6.1.2.1.

- *Memory consumption interfaces*

The memory manager retrieves and sends memory fragments that match the context descriptors supplied by a user to the memory consumption interfaces shown in Figure 40. They consist of a custom memory browser which is a new visual browser implemented in

this work; an HTTP browser which is a web browser on a desktop computer or mobile device such as a PDA; a mosaic which is a multi channel visual memory browser; and folders on a desktop computer or mobile device.

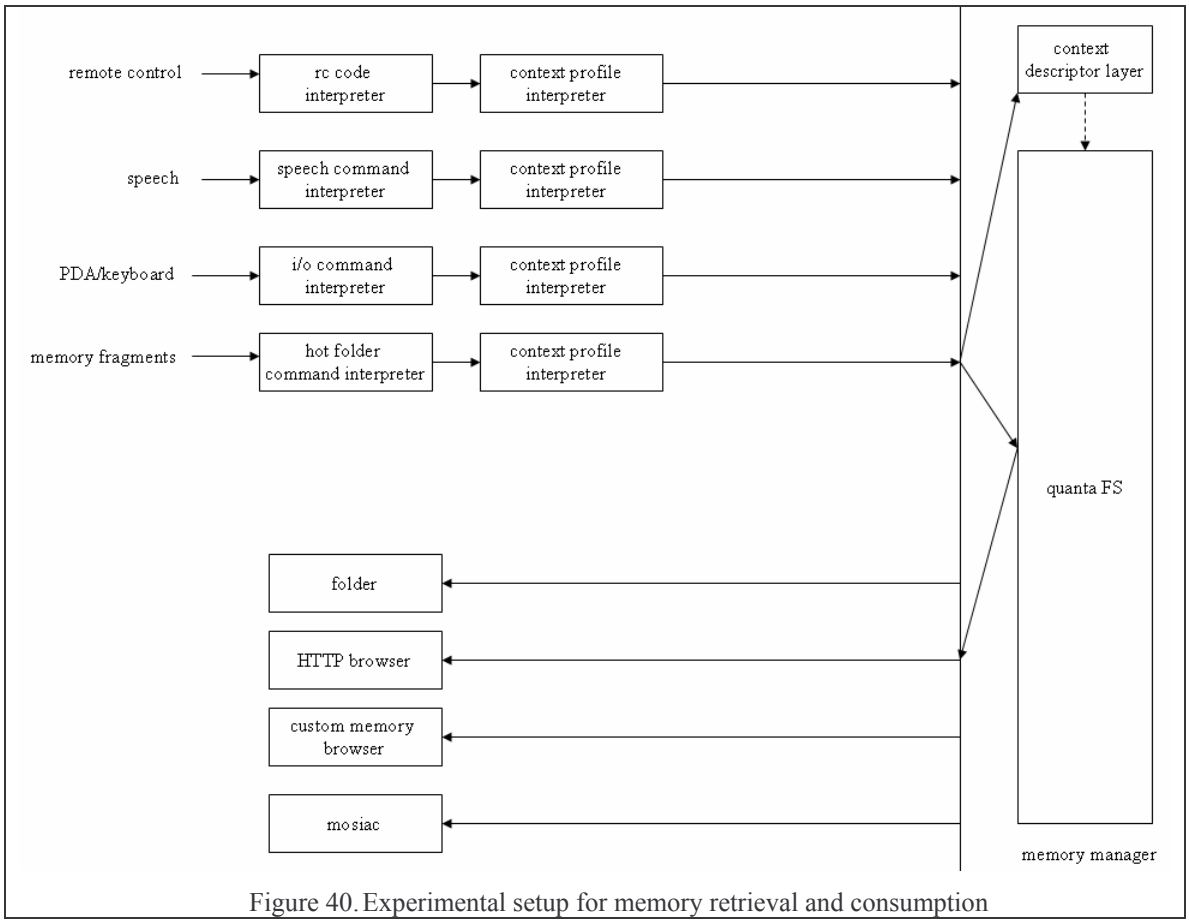


Figure 40. Experimental setup for memory retrieval and consumption

6.2 Hardware platform for the machine-based memory manager

The machine-based memory manager is described as consisting of memory producers, a memory manager and memory consumers in the model presented in chapter 4. Each of these entities is implemented as a combination of hardware and software components. The hardware platform of the machine-based memory manager consists of hardware for external memory capture, hardware that runs the software components of the machine-based memory manager and hardware for memory retrieval and consumption. The devices listed in Figure 41 have been assembled to construct the hardware platform. A distinguishing feature of the hardware platform is that it is packaged as a wearable

computer, as shown in Figure 42, so that it can be used everywhere. This section describes the role of each device in the machine-based memory manager (for design considerations prior to implementation refer to Appendix E).

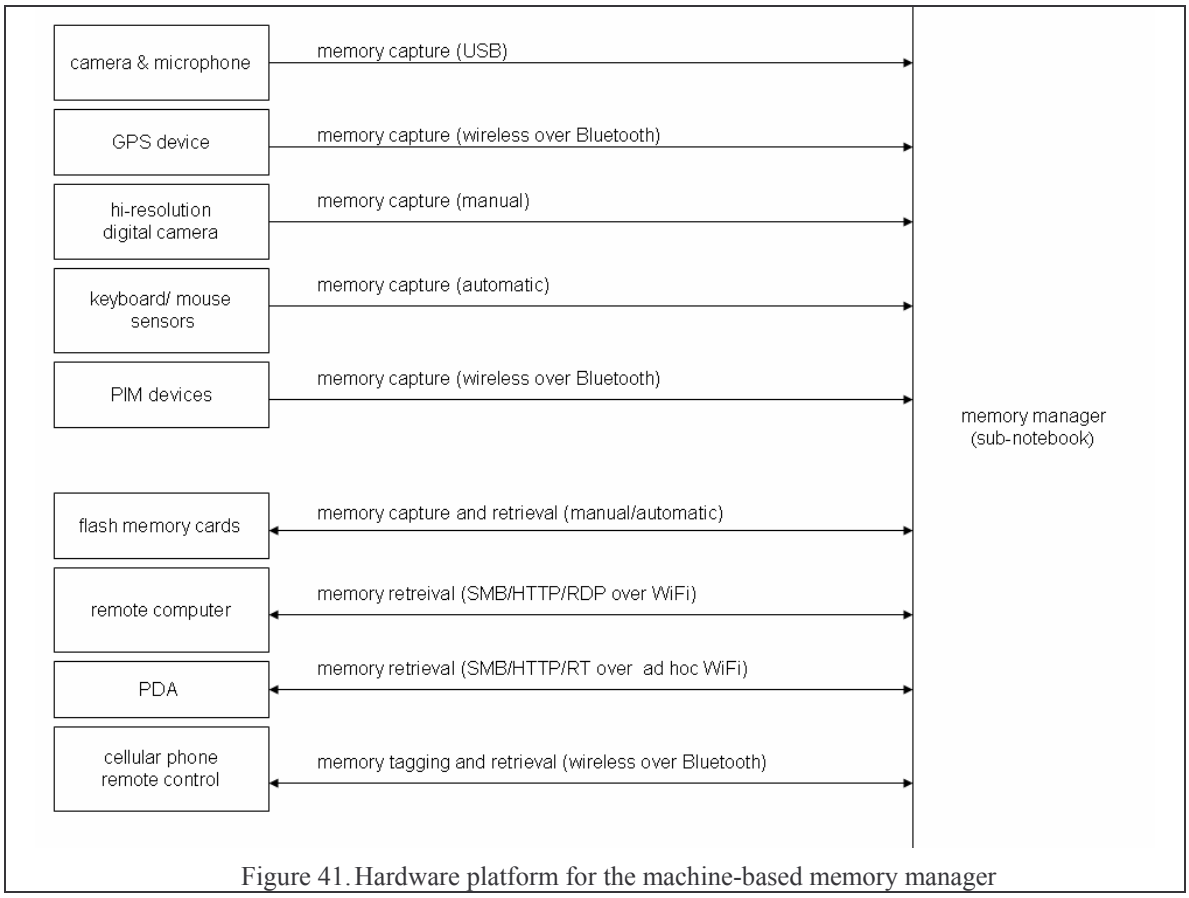


Figure 41. Hardware platform for the machine-based memory manager

6.2.1 Hardware for the machine-based personal memory manager

A sub-notebook computer (Dell X1) is used for running most of the software components of the machine-based memory manager. It contains a 1.1 MHz processor with 2MB of cache, 512MB of RAM and a 60 gigabyte hard drive. It also contains Bluetooth (low speed < 1Mbps) and WIFI (high speed ~54Mpbs) wireless network interface cards. The sub-notebook only weighs 1kg and has a small footprint in comparison to the size of a standard CD shown in Figure 42, which makes it easy to carry around. It is carried on the body by inclosing it inside a bag as shown in Figure 42. Thus, along with all the sensors attached to it, it is called a “wearable computer”.

6.2.2 External memory capture hardware

The machine-based memory manager contains a number of sensors that facilitate the capture of external memories. External memories can be analyzed by the memory manager for context descriptor extraction or they can be encoded into memory fragments for storage.

Figure 42. The memory manager and associated devices (from left to right) :webcam with original lens, webcam with fisheye lens, wearable in initial handbag format, wearable in backpack format, 5MP digital camera, sub-notebook, GPS device, PDA, cellular phone and wearable computer in backpack form while in use

6.2.2.1 Automatic external memory capture hardware

A number of sensors are used for automatic external memory capture by the machine-based memory manager.

1. A body worn video camera and microphone are used to capture video and audio sensor data. They are constructed by extracting the camera and microphone out of a desktop webcam made by Logitech [Logitech, 2005]. The default webcam lens is replaced by a fish eye lens which has a field of view of 180 degrees. This means that most of what a user sees can be captured. Figure 42 shows the camera before and

after the replacement of its default lens with the fish eye lens as well as the place where the camera is mounted. It streams video and audio data over a wired USB connection to the memory manager hardware.

2. A global positioning system (GPS) sensor is used to capture user location. It is implemented using a Socket Bluetooth GPS receiver [SocketCom, 2005] which streams user location information in longitude and latitude to the memory manager formatted as text strings (NMEA strings) over a wireless connection (Bluetooth over RFCOMM).
3. A multiple flash memory card reader device is attached to the memory manager which enables various types of flash memory cards that contain memory fragments to be used with the system. These memory fragments are absorbed by the memory manager through hot folders.
4. Sensors internal to the sub-notebook computer enable the memory manager to monitor user activity on a desktop computer, such as input device usage, the contents of a computer's screen and network activity.

6.2.2.2 Manual external memory capture hardware

A number of sensors are used for manual external memory capture by the machine-based memory manager.

1. A remote control is used for switching context profiles. It is implemented using a cellular phone (Nokia 6630) which is a device that can be operated with a single hand. It connects to the memory manager over a Bluetooth connection.
2. PIM devices such as cellular phones and PDAs can be used to send schedule and contact information to the memory manager over a Bluetooth connection.
3. A high resolution digital camera (5 Mega pixels) is used to capture video and images which have to then be transferred to the memory manager via the flash memory card reader described in the previous section.

6.2.3 Memory retrieval and consumption hardware

A number of memory retrieval and consumption devices are used by the memory manager.

1. A remote computer such as a laptop/desktop computer can interact with the memory manager over a wireless (WiFi) network connection using protocols such as HTTP [IETF, 1999], SMB [IETF, 1987] and RDP [Microsoft, 2005e].
2. A PDA device that contains a 320x240 resolution display and speakers, running the Microsoft Pocket PC operating system is used as a mobile memory retrieval device. Two hands are required to operate the PDA. The PDA is equipped with a Socket WIFI (11Mbps) network interface card [SocketCom, 2005b]. It connects to the memory manager over an ad-hoc wireless (WiFi) connection. The PDA can access memories from the memory manager using network protocols such as the HTTP, SMB and RT (older version of RDP).
3. The remote control described in section 6.2.2.2 which is used for manual external memory capture is also used for memory fragment retrieval. Memory fragments retrieved using the remote control are sent to a notebook/desktop computer for display or flash memory card for transportation.
4. Flash memory cards are used for memory retrieval as well as memory transportation. A particular flash memory card can be labelled with a context descriptor. When the memory card is inserted into the memory manager, memory fragments are transferred to/from that flash memory card.

6.2.4 Limitations of the hardware platform

The devices presented in the previous sections have a number of limitations. The sub-notebook has a limited amount of memory and processing power. The memory capture devices are manufactured by different organizations and capture external memories in various formats at different qualities. The memory retrieval and consumption devices are far from ideal as described in section 6.1.4. They do not provide natural methods of memory retrieval and consumption that people use every day, such as thought. Devices

which are part of the hardware platform such as the sub-notebook computer, GPS device and remote control require a constant supply of power which means that the machine-based memory manager requires some attention from a user in ensuring that all devices have adequate power.

6.3 Implementation of the software components of the machine-based personal memory manager

The instances of memory producers and memory consumers as well as the instance of the memory manager described in section 6.1, when implemented as software components, make up the software platform of the machine-based personal memory manager. This section describes the implementation of these software components as one or more black-boxes which are part of a graph model using the visual programming environment described in chapter 5. Each graph model represents a subsystem in the machine-based personal memory manager (for design considerations prior to implementation refer to Appendix E).

6.3.1 The memory manager

The memory manager consists of a context aware file system, a memory storage interface and a memory retrieval interface. The memory storage interface is implemented as a memory storage data channel which stores memory fragments and context descriptors in the context aware file system. The memory retrieval interface is implemented as a memory retrieval data channel which retrieves memory fragments from the context aware file system using context descriptors.

6.3.1.1 Implementation of the context aware file system

As described in chapter 4, the context aware file system (CAFS) consists of a context descriptor layer and a quanta file system layer. The context descriptor layer consists of information which is necessary to translate context descriptor states to quanta where memory fragments are stored (a context descriptor state addresses all memory fragments contained in quanta which are formed in the context described by that state). The quanta

file system layer consists of several quanta. Each quantum contains memory fragments as well as context descriptors which are formed in the same context as the memory fragments contained in a quantum.

In this implementation of the context aware file system, both the context descriptor layer and quanta file system layers are stored on an NTFS file system [Microsoft, 2001]. NTFS is used because it is optimized for storing a large number of files and enables fast access to stored files given a path identifier. File systems which have similar properties to NTFS such as Linux's ext2fs [Card, 1994] could have also been used. However, the machine-based memory manager currently runs on the Windows platform, therefore NTFS is used.

Design considerations prior to implementation

A number of different database architectures exist which are all candidates as an implementation medium for the CAFS. Hoffer [Hoffer et al, 1999] provides four categories of databases models: (1) hierarchical model, (2) networked model (3) relational model and (4) object oriented model. The CAFS inherently has a networked architecture. The memory fragments and context descriptors it manages can be conceptualised as objects. As a result we have chosen a database, which has a largely networked architecture but manages memory fragments and context descriptors as objects, as an implementation medium.

The advantages of a networked architecture are that:

- It can flexibly maintain relationships between context descriptors and memory fragments
- It can be highly optimised for a particular application (e.g. large volume, context based storage with consideration for high speed of access and reduced use of storage space)

The disadvantage of a networked architecture is that it can be complex to design and implement if an abstract representation of a file system such as the CAFS does not exist.

The chosen physical implementation medium for the CAFS (traditional hierarchical file system) has the following advantages:

- Hierarchical file systems are designed optimized for speed of access of nodes (files) on the system
- Hierarchical file systems are simple to work with. For example, each node on the hierarchical file system can be uniquely identified by a path.
- The chosen hierarchical file system (NTFS) is widely in use and has proven reliable for storing large amounts of digital data (subject to use of reliable hardware)

The implementation has the following disadvantages (which should be improved in future):

- Context descriptor states in the CAFS are not represented as complex objects (currently strings)
- No built-in mechanisms exist for insuring the integrity of data which is stored in the context descriptor layer or quanta file system layer, beyond those provided by the chosen file system.

For a further discussion of design considerations prior to implementation, which focus on the relation of the properties of the CAFS to its implementation, refer to Appendix E.

Context descriptor layer

In NTFS, information is stored in hierarchical format. As a result, the context descriptor layer consists of log files (one for every context descriptor state) which are stored in directories. In this implementation, a path such as */context descriptor/state/logfile* represents a log file. The log file contains a bit array that serves as a collection of quantum pointers. A bit array with a fixed size of 65kB holds quantum pointers for a 10 year period if the temporal size of a quantum is 10 minutes.

Quanta file system layer

The quanta file system consists of a collection of directories which are accessible using paths such as */year/month/day/hour/minute.* Each directory, which represents a quantum

of 10 minutes in length, contains memory fragments and context descriptor states. Memory fragments are stored as files, in the same format as they were received from memory producers. Context descriptor states are stored in log files, in a similar format to the log files in the context descriptor layer. The bit array inside each log file has a temporal length which is the same as that of a quantum. Each bit in array represents a second in the quantum. Thus, this quanta file system implementation has a memory fragment and context descriptor addressing resolution of one second even though the global quantum length is 10 minutes.

An alternative implementation of the quanta file system layer is provided on a SQL database for comparison with the NTFS based quanta file system. A SQL database is used because it is expected to provide faster retrieval times than NTFS. It also provides the SQL query language which NTFS lacks. SQLite [SQLite, 2005] and McDaniel's C# wrapper for SQLite [McDaniel, 2005] are used in the implementation. SQLite is a fast and portable (single file) relational database implementation with a claimed maximum database size of almost 2 Terabytes. In contrast, the maximum size of an NTFS partition is 2 ^ 64 bytes which is much larger than 2 Terabytes.

6.3.1.2 The memory storage interface

The memory storage interface, shown in Figure 43, enables memory producers to pass context descriptors or memory fragments to the memory manager for storage. Both of these items need to be accompanied with a time stamp that indicates the time of sampling from life streams.

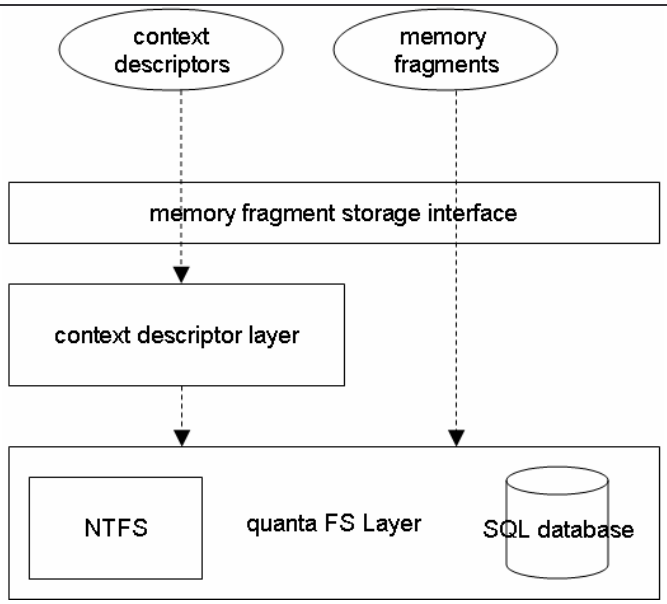


Figure 43. The memory manager's storage interface

This time stamp is used to locate the quantum, in the quanta file system, where memory fragments which are part of the current storage transaction will be stored. In the current implementation, a link to a memory fragment rather than the actual memory fragment is stored in a quantum, because copying memory fragments into quanta is, time-wise, an expensive operation. If the current storage transaction involves a context descriptor, the time stamp is used to update a context descriptor's entry (log file) in the context descriptor layer of the CAFS as well as its entry inside a quantum.

6.3.1.3 The memory retrieval interface

The memory manager's memory fragment retrieval interface is shown in Figure 44. In order to retrieve memories, memory consumers need to present one or more context descriptors to the memory manager. On receipt of context descriptors, the memory manager performs memory retrieval in two stages: (1) it maps context descriptors to quanta and (2) it retrieves the contents of quanta and returns them to the memory consumer that initiated the memory retrieval transaction.

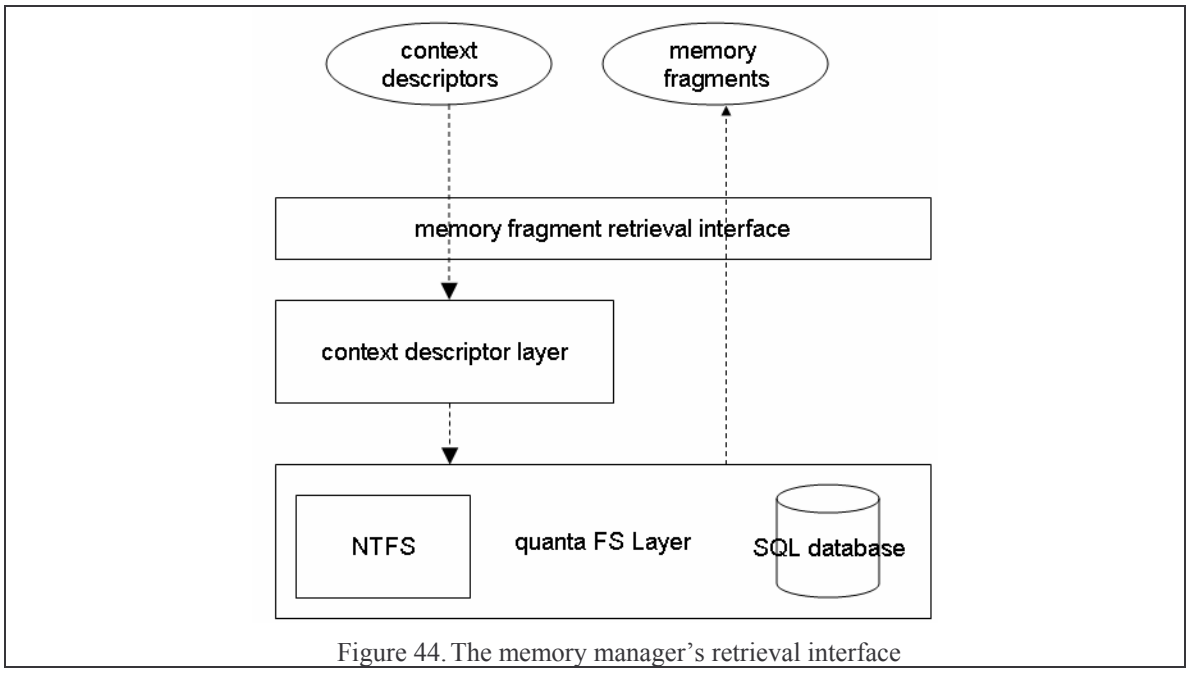


Figure 44. The memory manager's retrieval interface

Memory fragment retrieval stage 1: mapping context descriptors to quanta

The memory manager examines the context descriptor layer of its context aware file system and creates a set of quanta pointers for each context descriptor which is matched to those supplied by a memory consumer. Using these sets of quanta pointers, an intersection operation is performed in order to identify a single set of quanta pointers which is associated to all the context descriptors supplied by a memory consumer. This set of quanta pointers is called the *global retrieval set*.

Memory fragment retrieval stage 2: retrieving memory fragments from quanta and returning them to memory consumers

During the second stage, the memory manager examines the contents of quanta which are in the global retrieval set. The maximum number of quanta which are examined from this set is determined by the memory consumer which initiated the memory retrieval request through the memory retrieval specifier keyword "maxitems" (described in section 6.3.3). During the examination of each quantum, an intersection operation which involves the contents of the log file of each context descriptor which is contained in a quantum is performed to create a second retrieval set called the *local retrieval set.* The local retrieval set applies only to the current quantum and temporally identifies which memory

fragments in a quantum match the specified context descriptors. Memory fragments whose creation time matches the times in this set are retrieved from a quantum. The maximum number of items which are retrieved from a quantum is determined through the memory retrieval specifier keyword “itemsperquanta” (described in section 6.3.3).

During stage 2 of memory retrieval, there is also an opportunity to retrieve memory fragments from databases which are external to the memory manager. In this implementation, accessing of Google’s Desktop Search database [Google, 2005], [Bayons, 2005] was considered. However, Google’s desktop indexer, which is currently in beta testing, does not allow retrieval using time intervals constructed from quanta pointers even though the Google API documentation [Google, 2005b] states that time based retrieval is supported. Other possibilities for external databases exist, for example, MSN’s desktop search’s index [Microsoft, 2005] as well as numerous other online databases.

6.3.1.4 Portability and security

The memory manager used in this system is designed to be autonomous and secure. The data managed by the memory manager, which consists of a context aware file system, is stored in a single file that is encrypted (blowfish algorithm, 448 bit encryption key) using the TrueCrypt virtual volume and encryption system [TrueCrypt, 2005]. This makes the file system secure, portable and easy to back up. TrueCrypt reports that the drive has an average performance of 20MB/second in encrypted mode, thus the drive is reasonably fast even in encrypted mode.

Even though the context aware file system is created on the Windows platform, the data which is stored in the CAFS is accessible from other operating systems such as Linux.

6.3.2 Automatic context descriptor extraction and memory fragment capture software

Memory producers have been identified in chapter 4 as entities that capture memory fragments and identify context descriptors from life streams. This section presents the implementation of memory producers that perform their function automatically using the software environment's data flow based modelling environment. The memory producers are grouped into five categories, presented in sections 6.3.2.1 to 6.3.2.5.

6.3.2.1 Context descriptors and memory fragments from video

Memory producers which capture and analyze video data are represented by the graph model shown in Figure 45. The graph model enables the extraction of context descriptors that indicate the number of faces, text from video, light level, level of motion in video and number of objects in video. The graph model also captures a video stream, video frames containing faces and video frames containing motion as memory fragments.

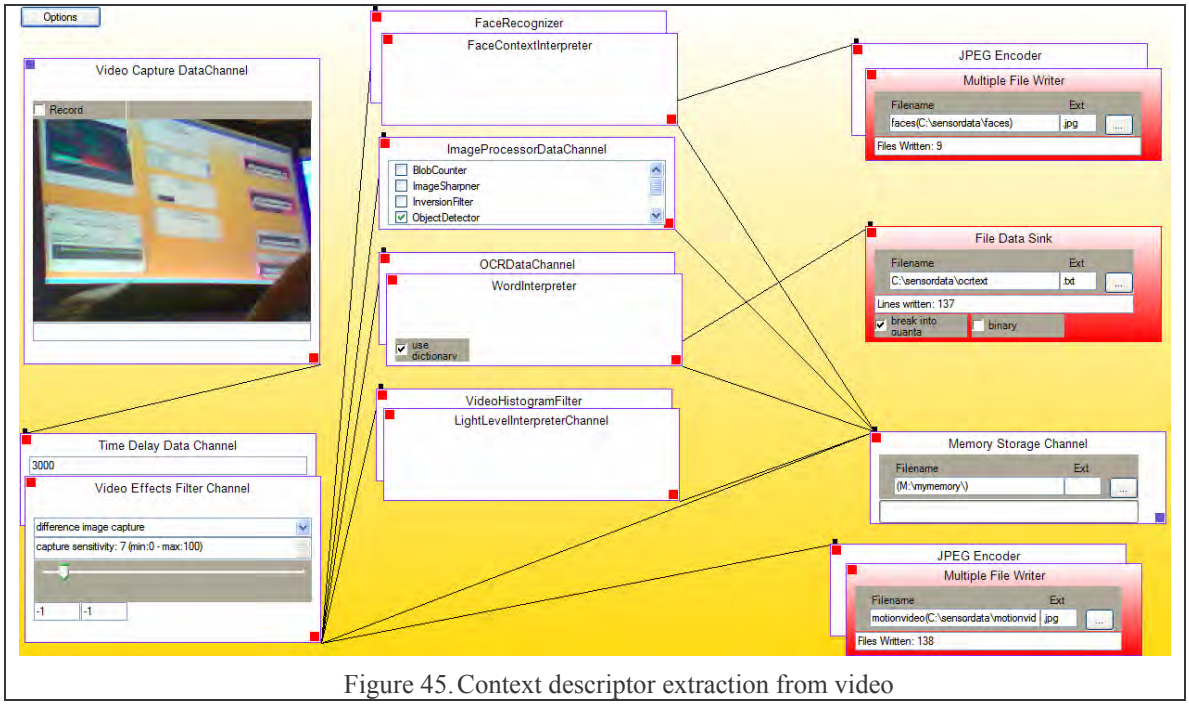


Figure 45. Context descriptor extraction from video

Level of activity in video

The level of activity in video is identified by the video effects filter data channel, shown in Figure 45, which can perform a variety of video related operations. In this graph model, it performs motion detection. For every video frame passed to it by the video capture data channel, it outputs a context descriptor for the level of activity in the current video frame (either low, medium or high) which is pushed to the memory storage data channel. If the level of activity exceeds a threshold value which in this implementation is set at 7%, the motion detector also outputs a video frame. This video frame is passed to other video context descriptors for further processing and is also saved by the multiple file writer data sink after being encoded by a JPEG encoder data channel.

Number of faces in video

The number of faces in video frames is identified by the face recognizer data channel and the face context interpreter data channel shown in Figure 45. The face recognizer data channel processes each video frame passed to it by the motion detector and outputs an image containing faces and a context descriptor for the number of faces detected (either one, two, three or multiple). The context descriptor is pushed to the memory storage data channel and images containing faces are saved by the multiple file writer data channel after being encoded by a JPEG encoder data channel. The face context interpreter could do additional work and identify people to whom faces belong to, however, extraction of the number of faces in video is considered sufficient for this implementation. The face recognizer is implemented using the open computer vision library (OpenCV) [Intel, 2005] and Wentworth's [Wentworth *et al*, 2003] C# wrapper for OpenCV.

Text from video

Text contained in video frames is identified by the OCR (optical character recognizer) data channel shown in Figure 45. The OCR data channel processes each video frame passed to it by the motion detector and extracts text contained in the frame. The word context interpreter looks up text identified by the OCR in a dictionary and, if the text is found in the dictionary, outputs an OCR context descriptor in the form (word1, word2, …, wordn) which is pushed to the memory storage data channel. Raw text is pushed to

the file data sink which breaks data along quanta boundaries. The OCR is implemented using the Microsoft office OCR library [MODI, 2005]. The dictionary used by the word interpreter is implemented using the NetSpell library [Welter, 2005].

Light level from video frame

Light level in video frames is identified by the video histogram filter data channel and light level context interpreter shown in Figure 45. The video histogram filter data channel processes each frame passed to it by the video data channel and builds a brightness histogram of each captured frame The light level interpreter channel examines the brightness histogram passed to it by the video histogram filter data channel and generates context descriptors which indicate the brightness of a video frame (either darkness, lowlight or visible). These context descriptors are then pushed to the memory storage data channel. Pixel brightness values returned by *C#'s Color.GetBrightness()* method are used.

Number of objects in video

The number of objects in video is identified by the image processor data channel shown in Figure 45. The data channel performs object detection by applying an edge detection filter, binarization filter and blob counter filters on every image passed to it by the motion detector. The edge detector identifies the edges of objects and the binarization filter converts the image containing identified edges into a monochrome image. The blob counter counts the number of groups of white objects in a monochrome image. In this way, a rough count of the number of objects in a video can be determined. The object detector outputs values one, two, three or multiple to indicate the current number of objects in a video frame. The three image processing filters are implemented using Kirillov's [Kirillov, 2005] image processing library.

Continuous capture and archiving of video streams

The video capture data channel, shown in Figure 45, captures video frames for context descriptor extraction. It can also continuously record and archive video streams. It is impractical to store a very long stream of video into a single data file given current storage media which provide slow read/write time when accessing large files. Therefore,

the video capture data channel breaks recorded video streams along quanta boundaries for convenient storage in the context aware file system. Captured video is encoded using XviD video codec [XviD, 2005] at 1Mbps, 320x240 and mp3 audio codec at 64Kbps, mono (A recent benchmark [doom9, 2005] shows that XviD is one of the best low bit rate video encoders currently available). The video capture data channel can resume/pause recording when the camera device which is in use is plugged/unplugged. Low's C# DirectShow library [Low, 2003] is used to implement the video capture data channel.

A second method of capturing continuous video is also implemented using VLC [VideoLan, 2005]. VLC can record video from a number of video sources such as a camera, network or TV card using the AVC video codec which the doom9 benchmark shows is even better than XviD. The remote control application shown in Figure 46 enables the use of VLC for perpetual video recording in a similar fashion as the video capture data channel.

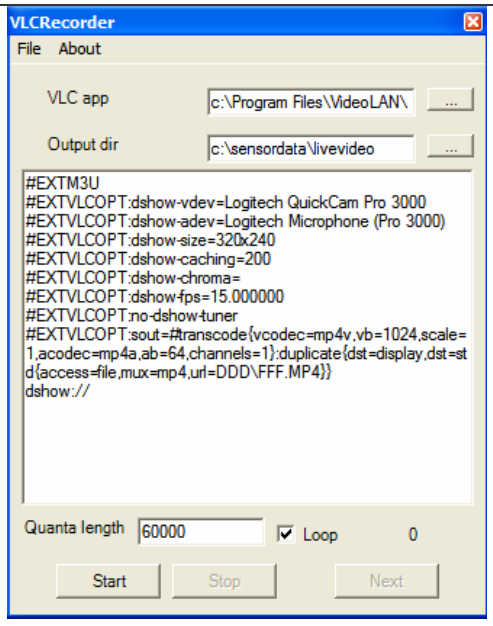


Figure 46. VLC remote control application

6.3.2.2 Context descriptors and memory fragments from audio

Memory producers which capture and analyze audio data are represented by the graph model shown in Figure 47. They enable the extraction of sound type and speech context descriptors. They also capture audio and text in speech as memory fragments.

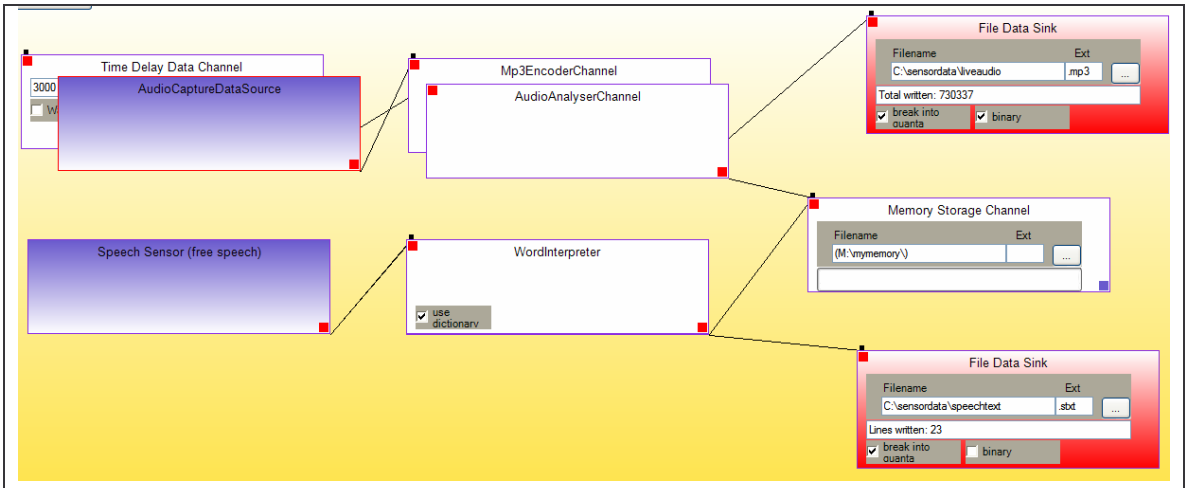


Figure 47. Context descriptor extraction from audio

Noise type

The audio analyzer data channel shown in Figure 47 identifies different types of noises by analysing audio samples passed to it by the audio capture data source. Samples are captured at 8 bits, 11025 Hz, single channel. The audio analyzer data channel monitors changes in the amplitude of captured audio. Since the audio is captured at 8 bit resolution the amplitude of the sound wave is in the range 0 to 255 (2^8). The audio analyzer produces context descriptors that indicate the noise type (silence, noise, constant noise, loud noise, constant loud noise) which are pushed to the memory storage channel. Raw audio is encoded through the mp3 encoder data channel and saved by the file data sink which breaks data along quanta boundaries. Munoz's C# audio capture library [Munoz, 2003], Cardoso's [Cardoso, 2004] C# mp3 encoding library and Lame mp3 encoder [LameEncoder, 2004] are used.

Speech

Speech is identified by the speech sensor shown in Figure 47. The speech sensor converts speech to text using the Microsoft speech to text (STT) engine [Microsoft Speech, 2005].

The engine fires the *OnRecognized* event when speech is recognized and the *OnHypothesis* event whenever any text is recognized even if it is not a valid word. Thus while the *OnRecognized* event can be used to obtain context descriptors from the text in speech, the *OnHypothesis* event indicates speech is occurring (serves as a source of sound type context descriptor). The word interpreter checks recognized text against a dictionary. It then outputs a speech context descriptor in the form (word1, word2, …, wordn) and (indictionary) if the speech contains words which are in the dictionary. It also outputs a sound type context descriptor for speech. The full recognized text is also saved to a log file which is broken down along quanta boundaries by the file data sink.

6.3.2.3 Context descriptors and memory fragments from GPS data and PIM information

Memory producers which capture and analyze data from a GPS sensor and PIM information sources are represented by the graph model shown in Figure 48. The graph model enables the extraction of location, people and event context descriptors. It also captures location sensor data as well as contact and schedule entries from PIM devices as memory fragments.

Location

Location is identified by the location context interpreter shown in Figure 48. GPS data formatted as NMEA strings [Baddeley, 2005] is pushed by the serial port sensor (which is connected to a GPS device) to the location context interpreter and a file data sink. The location context interpreter extracts longitude and latitude information from GPS data and translates them to real location context descriptors such as home, office, library etc. The file data sink stores raw GPS data to a log file. Serial port IO is performed using Krell's C# library [Krell, 2003].

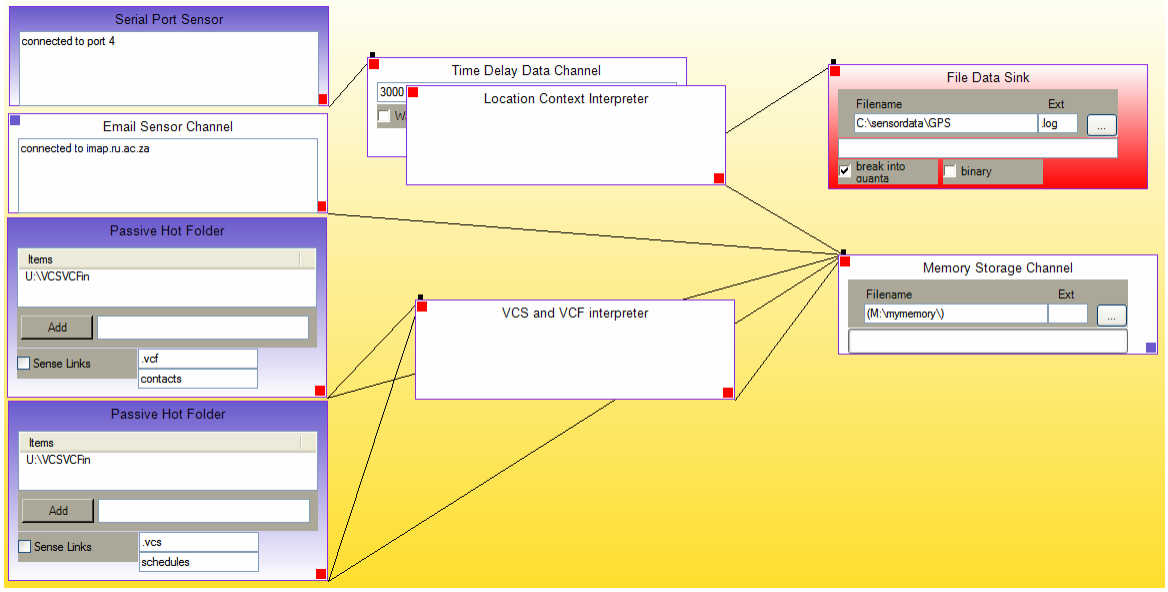


Figure 48. Context descriptors extraction from GPS and PIM information sensors

Email

Context descriptors are extracted from email messages by the email sensor data channel shown in Figure 48. It monitors an individual's email inbox and identifies the recipient, sender and subject of an email as context descriptors. The file data sink saves email messages. The email sensor used here is implemented using LumiSofts's C# email client library [LumiSoft, 2003] and Mentalis' C# SSL library [Mentalis, 2003].

People and events

People and event context descriptors are identified by the VCS/VCF interpreter data channel shown in Figure 48. VCS/VCF encoded files are produced by PIM applications such as Outlook and PIM devices such as cellular phones. A VCS file contains a calendar item definition that contains an event description, event start and end times. A VCF file contains a contact item definition in address book format (person's name, phone number etc.). Individuals can create and send VCS files to their machine-based memory manager using their PIM device's calendar application. The VCS/VCF interpreter data channel processes VCS/VCF files and produces context descriptors that indicate a person's name or an event description. A hot folder is used to monitor the contents of a folder for incoming files in VCS/VCF format.

6.3.2.4 Context descriptors and memory fragments from a desktop computer

Memory producers which capture and analyze data from a desktop computer are represented by the graph model shown in Figure 49. The graph model enables the extraction of context descriptors for the active application, all running processes, input device usage, words which are typed, and visited web pages. It also captures the contents of the desktop screen, all text typed by a user and web pages as memory fragments.

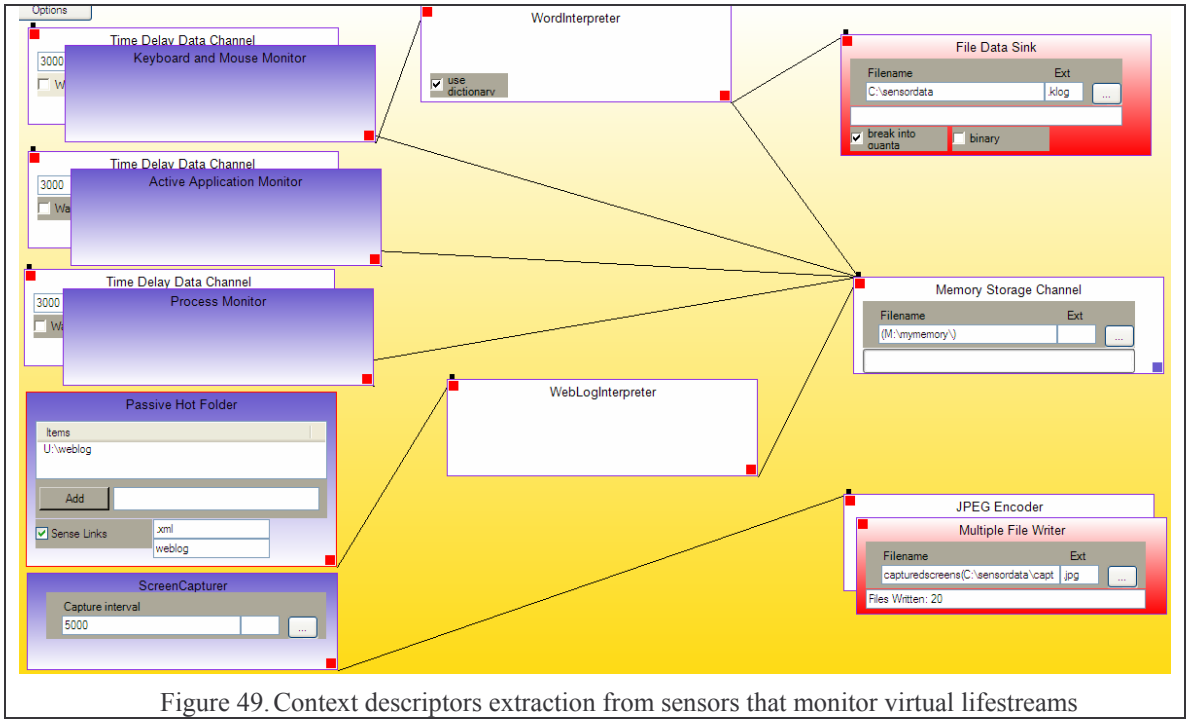


Figure 49. Context descriptors extraction from sensors that monitor virtual lifestreams

The active application

The active application context descriptor is identified by the active application monitor data source shown in Figure 49. It outputs context descriptors for the current application name and window title. The active application can indicate what activity or task a user is currently involved in. For example, if a user is interacting with a web browser, then their current activity is browsing the web and, if they are interacting with a word processor, then their current activity is document editing. The active application is determined through the *GetActiveApplication()* and *GetModuleFileName()* Win32 API calls. The active application title is determined through the *GetWindowText()* Win32 API call.

All running processes

Context descriptors for all applications running on a computer are identified by the process monitor data source shown in Figure 49. Running processes are an important context descriptor. They indicate what task/activities a user is engaged in. The process context interpreter generates a context descriptor that consists of process names.

Input device usage

Input device usage context descriptors are identified by the keyboard monitor data source and word interpreter shown in Figure 49. The keyboard and mouse monitor outputs context descriptors that represent key presses and mouse movements. The word interpreter checks typed words against a dictionary and produces as context descriptors each typed word as well as whether or not the words are in the dictionary. It also outputs sentences which are saved by the file data sink. The keyboard and mouse monitor is implemented by installing a global mouse and keyboard event listener on the machine being monitored using the win32 *SetWindowsHookEx()* API call. A global hook captures keystrokes and mouse movements from all applications.

Visited web pages

Context descriptors that indicate the web page host and title visited by an individual are extracted by the weblog interpreter shown in Figure 49. A hot folder is used to monitor web pages that are saved to a folder by Slogger [Schutte, 2005] which is an extension for the Mozilla Firefox web browser. It captures visited web pages in their entirety as memory fragments.

Periodic screen capture

The screen capture data channel shown in Figure 49 captures an image of a user's screen periodically as a memory fragment. Capturing desktop screens periodically enables individuals to visually inspect and retrieve information from their past computer screens. The screen capture data channel grabs an image every 5 seconds at the desktop's resolution of 1280x1024. Captured images can also be processed by an OCR data channel to extract text information, but this operation is CPU intensive due to the high

resolution of the captured image. Scaling the image to a smaller dimension, such as 320x240 or 640x480, loses much of the text information contained in captured images.

6.3.2.5 Context descriptors and memory fragments from hot folders

The concept of hot folders is explained in section 6.1. Hot folders can be labelled to represent any context descriptors. Memory fragments which are placed in them are automatically absorbed and passed to the memory storage data channel. Hot folders can also use a disk drive's volume label as a context descriptor if the current context descriptor is set to *#usedrivelabel*. In addition to a hot folder's label, metadata extracted from memory fragments absorbed by a hot folder serve as context descriptors. Memory producers which consist of hot folders are shown in Figure 51.

Hot folder implementation

Two types of hot folders are utilized in this implementation: an active hot folder and a passive hot folder. An active hot folder uses a worker thread to periodically scan folders that a particular hot folder is assigned to watch. This involves getting a listing of files from folders and checking if new files have arrived since the last scan. In comparison, a passive hot folder does not perform an active scan of monitored directories. Instead, a file system event handler routine is registered using the C# *FileSystemWatcher* object which enables the event handler to be called by the operating system whenever memory fragments are placed in a folder, removed from a folder, or are modified inside a folder. Active hot folders have the advantage that the hot folder never misses new incoming memory fragments since it performs periodic scans. It can also be used to scan memory fragments created in the past. Passive hot folders have the advantage that no additional work is necessary in keeping track of memory fragments since the operating system notifies the hot folder when changes occur. However, if a passive hot folder is not active it can miss incoming files. The *FileSystemWatcher* API only works for hot folders which exist on a file system. It does not support notification when a mobile disk is added/removed or the root hot folder is created/deleted. Active hot folders overcome these limitations.

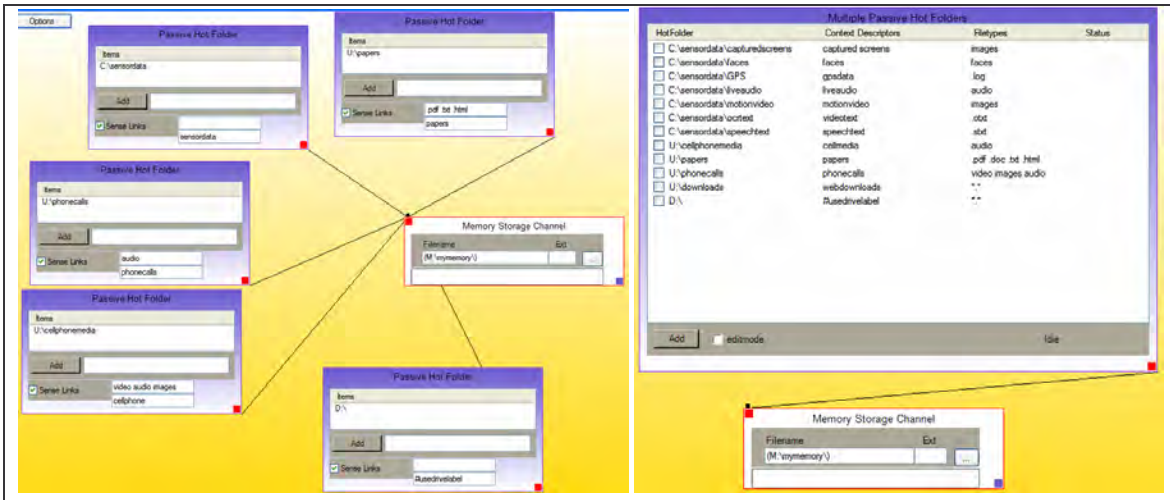


Figure 50. Context descriptors from hot folders (single and multiple hot folders)

Metadata extraction

When it receives memory fragments from hot folders, the memory manager uses a helper process called *MetadataExtractor,* which is a separate application from the memory manager, to extract metadata for use as additional context descriptors. Because *MetadataExtractor* is separate from the memory manager, it can be extended without modifying the memory manager. *MetadataExtractor* currently extracts various types of metadata, including document size, name, author, type; for video files - frame width, height, video length and for audio files - length and author information. Extracting metadata from multimedia files can be challenging because several multimedia data encoding formats (codecs) exist. For extracting metadata from such files, three combinations of applications are used: (1) Mplayer [Mplayer, 2005] which is an open source media player with support for many multimedia codecs, (2) FFmpeg [FFmpeg, 2005] which is a multimedia file encoder similar to Mplayer and (3) Directshow enhanced with ffdshow [ffdshow, 2005], which is a collection of open source directshow filters. These three applications are used in that order. Mplayer is the most robust metadata extraction application, as it is able to processes several types of multimedia files including corrupted ones. These applications are also used to extract thumbnails from video memory fragments which help users preview items during memory retrieval.

6.3.3 Manual context descriptor extraction software

Two methods of manual context descriptor extraction (speech and remote control based) are introduced in section 6.1. This section presents graph models which represent their implementation. Both methods use a context interpreter called a user query filter which converts user queries to context descriptors. Its implementation is also described here.

6.3.3.1 User query filter

The user query filter enables the mapping of a number of queries to context descriptor states which can be used by the memory manager for memory fragment retrieval. The user query filter, which is implemented using the C# regular expression parser, can recognize temporal queries, context descriptors, context profiles, memory fragment retrieval specifiers and saved queries.

Temporal queries

Temporal queries specify the time interval, containing quanta, from which memory fragments should be retrieved. Available temporal keywords include: “all” which means the time interval containing all quanta , “now” which means the current quanta, today, yesterday, tomorrow, last/this/next week, last/this/next month, last/this/next weekday (where weekday is Monday to Friday), last/next/this monthname (where monthname is January to December), date st-et (where st and et use the format DD/MM/YYYY:H:M:S and st-et specifies a time interval) and date time (where time uses the format DD/MM/YYYY:H:M:S and represents some quantum).

Context descriptors

Memory consumers can retrieve memory fragments by directly supplying context descriptors. The format of context descriptor key words that the user query filter understands is $\{Cd_i:cd_is_1,cd_is_2,..,cd_is_n\}$, where Cd_i is a context descriptor and cd_is_n are its state. If a query of form “cd_is_1 cd_is_2” is presented to the user query filter, it will attempt to map the specified states to Cd_i, but this mapping is not guaranteed to be correct all the time because two or more context descriptors can have the same state descriptor. For

example consider *light level (low)* and *noise level (low)*. To accurately describe each context descriptor the query format {light level:low} {noise level: low} should be used.

Context profiles

The user query filter expects context profiles of form {profile:profile1, profile2,…, profilen} where profilen is a particular context profile definition. The profile tag indicates to the user query filter that the current query item is a profile and should be decomposed to context descriptors.

Memory fragment retrieval specifiers

Three special keywords are recognized by the user query filter as descriptors of how in relation to a quanta, memory fragments should be retrieved and the amount of memory fragments which are to be retrieved per query. The keywords “maxitems=n” or “mi=n” indicate to the memory manager that a memory consumer is interested in receiving at most *n* memory fragments. The keywords “itemsperquanta=n” or “ipq=n” specify that at most *n* items should be retrieved from each quantum that matches a particular query. The keyword “matchingdistance=n” or “md=n” indicates that during memory fragment retrieval from a quantum the temporal distance of a memory fragment from a context descriptor state should be *n* seconds. A small value for *n* returns the memory fragments which are temporally closest to a context descriptor state.

Saved queries

A saved query is a mechanism of combining all of the above forms of queries into a single query. It enables the construction of a simple query that represents a complex query. A saved query can contain temporal queries, context descriptors and context profiles. Simple queries that map to complex queries are, for example, used extensively by search engines such as www.yubnub.org.

6.3.3.2 Context descriptors from speech

The memory producer which identifies manually supplied context descriptors from speech is shown in Figure 52. It consists of a speech based memory tagger data source

and user query filter data channel. The speech based memory tagger is implemented using a speech to text (STT) engine with a custom grammar. Defining a custom grammar increases the accuracy of the STT engine. The grammar, in this case, consists of context descriptors and their states as well as context profile descriptors. Individuals can change context descriptors by issuing speech commands which consist of context descriptors and profiles such as “person Alfredo”, “location home”, “profile dining” etc. To enable individuals to edit the current context state before it is saved, a command stack is necessary. The command stack is manipulated using a number of speech commands:

1. command:new state - commences the composition of a new context state
2. command:save - makes the currently composed context state the active one
3. command:reset - clears the context state which is being edited. A reset command followed by a save command clears the current context state
4. command:current state - displays the current context state to memory consumers if they need to know what the current state is

Manipulating the command stack is not necessary when effecting context state changes through context profiles. The command stack is cleared, the current context state set to the selected context profile and the command stack saved automatically.

6.3.3.3 Context descriptors from a remote control

The memory producer which extracts manually supplied context from a remote control is represented by the graph model shown in Figure 52. It consists of a serial port data channel which interfaces to a remote control, remote control command interpreter data channel, profile manager data channel, user query filter data channel and periodic context descriptor generator data channel. The remote control is implemented using the Nokia 6630 cellular phone which is shown in Figure 51. A J2ME midlet (java application for mobile devices) [Sun, 2005] called Life Terminal which is based on the Bluetooth serial terminal application developed by O'Sullivan [O'Sullivan, 2005] is implemented and used to stream button presses from the phone to the machine-based memory manager wirelessly (RFCOM over Bluetooth). Life Terminal is shown in Figure 51.

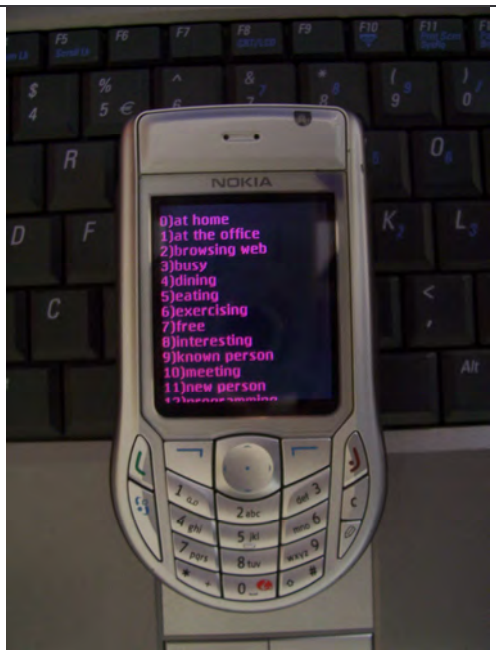


Figure 51. Life Terminal, a J2ME remote control applicaion for context profile switching

In Life Terminal, a list of available context profiles is displayed on the cellular phone's display. The subject switches profiles by entering commands such as:

0# to select context profile 0
1# to select context profile 1
n# to select context profile n
**# to display the current profile*
***# to list all available context profiles*
*n***# to list all available context profile starting from profile n*
****# to clear the current context profile state*

The serial port data channel, shown in Figure 52, pushes incoming remote control key presses to the command interpreter which identifies Life Terminal commands and pushes them to the profile manager.

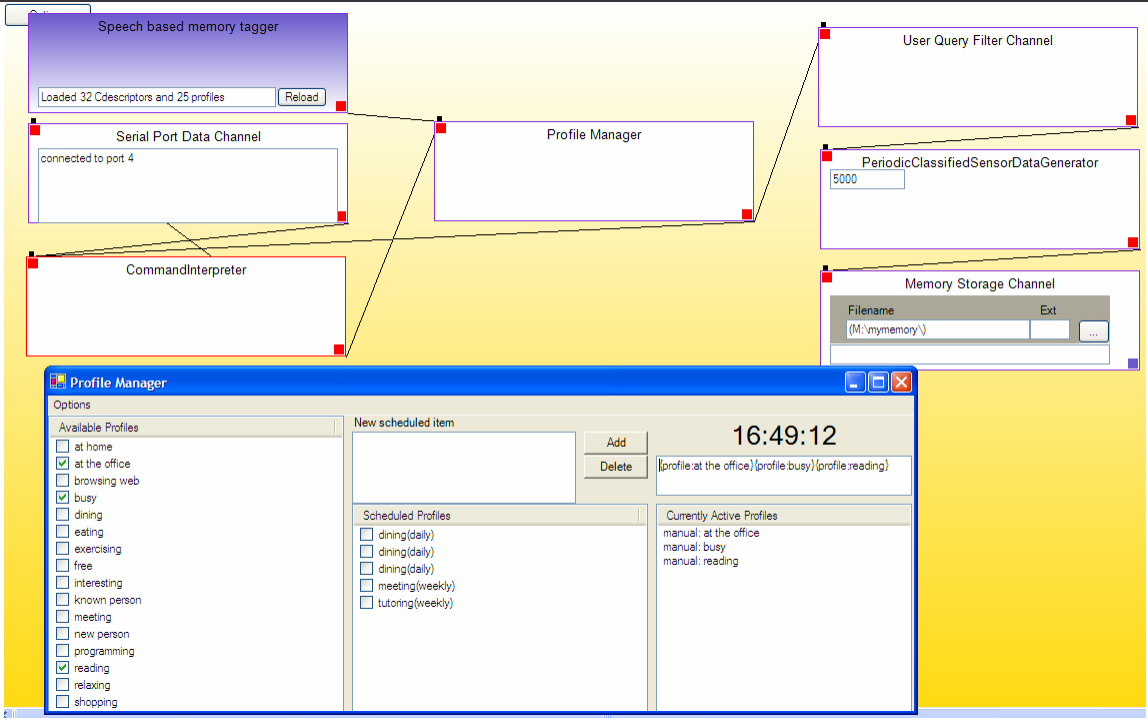


Figure 52. Graph model for context profile switching with speech, a remote control or GUI

The profile manager offers the GUI shown in Figure 52 that shows available context profiles, currently active context profiles and enables scheduling time based context profile changes. As an individual switches context profiles with a remote control, he can observe the changes on the profile manager's GUI or his cellular phone. When a change occurs, the profile manager pushes the currently selected context profile to its sink, which in this case is a user query filter data channel. The periodic context descriptor generator periodically pushes currently selected context descriptors to the memory storage data channel.

6.3.4 Memory retrieval and consumption software

The memory fragments managed by the memory manager will ultimately have to be consumed by individuals. Section 6.1 has identified a number of memory retrieval and consumption interfaces. They consist of interfaces for accepting queries from individuals and interfaces for presenting retrieved memory fragments to individuals. This section presents the implementation of each of these interfaces.

6.3.4.1 Interfaces for accepting queries from people

Four types of interfaces are implemented for accepting queries from people. They are: remote control, speech, PDA/Keyboard and hot folders.

Remote control based queries

Implementation of a remote control based interface is considered in section 6.3.3.3 where it is used for manual context descriptor extraction. A different set of commands is used for memory retrieval. A remote control command map file is used to define a mapping from remote control codes to context descriptors or context profiles. An example remote control key map file looks as follows:

50# ->video clip now
51# ->audio clip from now
52# ->screen capture from now
60#->last phone call
100#->interesting things from today
101# -> today videos
102# -> yesterday videos
103# ->last week videos
200#->meeting last week
201#->tutoring last week
300#->next lecture
600# ->shopping list
555#->next event
556# ->events this week
700#->new emails
800#->launch word processor and document "thesis" "chapter1"
801#->launch web browser
3000#-> paper this year "context awareness"
4000#-> relaxing music

Hence, to retrieve memory fragments of a particular type, all an individual has to do is to enter a number such as 100#, 101# etc. and matching memories are retrieved. The key code 555#, for example, returns a memory fragment that describes the closest future event. The remote control based memory retrieval interface uses the "channel metaphor" for retrieving memories. The channel metaphor is a concept which everyone that uses a television is familiar with (the importance of this concept was realized in this work after encountering Jay's MeTv [Jay, 2005]). When watching television, people access "memories" of a particular type by changing channels. For example, if a user knows that there is a news channel broadcasting on channel 51, he expects to receive memory fragments related to news whenever the user selects channel 51. Remote control based

memory retrieval exploits this. After relating queries to channels identified by numbers, users can become accustomed to consuming particular types of memory fragments when they punch in a code in their remote control. Just like cable TV stations provide thousands of TV channels to consumers, the memory manager can supply thousands of channels of personal memories to an individual.

Speech based queries

Speech based memory retrieval is similar to the remote control based retrieval, except that users describe what they want through speech. Its advantage is that users do not have to use remote control codes. Its disadvantage is that speech is not appropriate for use in every environment.

PDA/Keyboard based queries

Keyboard/PDA based interfaces involve the use of a keyboard and/or stylus which are the most common form of interacting with computers today. They are implemented through input devices sensors that exist on computers/PDAs. Users describe their context through textual context descriptors which they type.

Hot folder based queries

Hot folders enable queries in the form of memory fragments rather than textual context descriptors. Users place the memory fragments that they desire to use as retrieval cues inside a hot folder, which passes the memory fragments to a context interpreter that extracts context descriptors which will be used for memory fragment retrieval. The memory producers described in section 6.3.2 can be used for this purpose.

6.3.4.2 Interfaces for presenting memory fragments to people

A number of interfaces are used in the experimental system for presenting memory fragments to an individual and allowing him to transfer them to other people. They consist of a custom memory browser, an HTTP browser, folders and a mosaic.

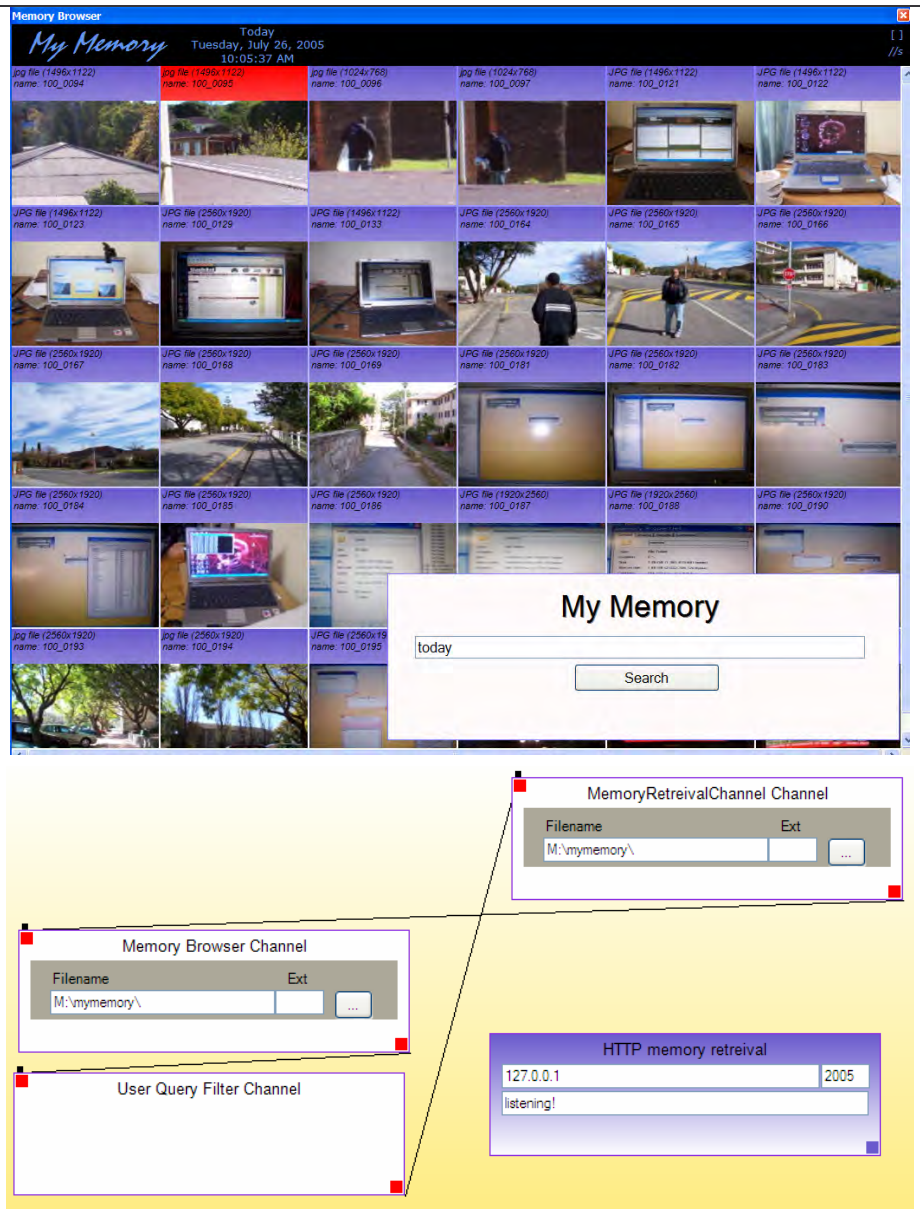


Figure 53. Custom memory browser and HTTP based memory browser channel

Custom memory browser

The custom memory browser is a browser implemented during this work and is shown in Figure 53. It can accept input from any of the input interfaces described in section 6.3.4.1. It presents memory fragments as colour coded thumbnails. For example, orange for video, purple for images, green for audio etc. Individuals can select groups of memories and transfer them to other people. They can also manually re-tag selected memory fragments with context descriptors. Individuals can consume memory fragments using the operating system's default memory fragment renderer.

Figure 54. PDA based memory retrieval interface showing live video from the wearable camera using [CoreCodec, 2005] and the onboard PDA web browser being used to retrieve memories from the memory manager through the HTTP based memory browser

The graph model that represents the custom memory browser is shown in Figure 53. It consists of a memory browser channel which implements the visual part of the memory browser, a user query data channel and a memory retrieval data channel which represents the retrieval interface of the memory manager.

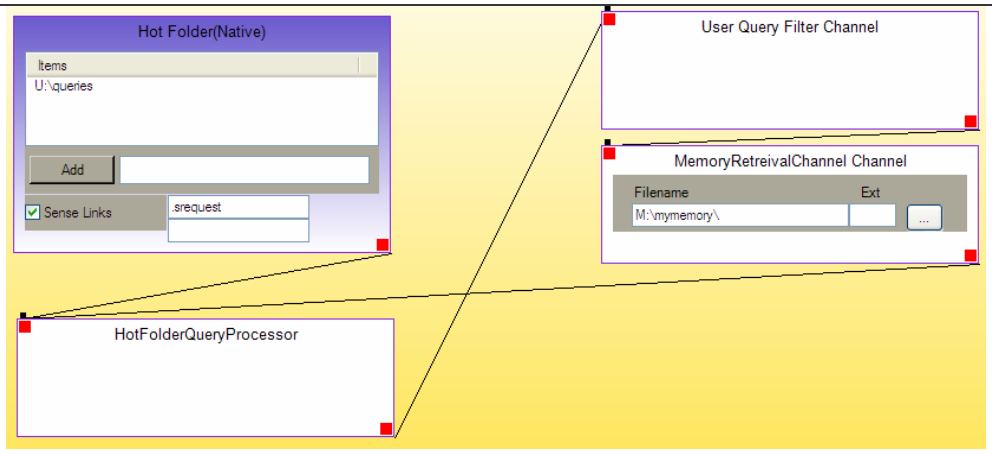


Figure 55. Hot folder based memory browser

HTTP based memory browser

The HTTP based memory browser is a memory consumer implemented as a server that accepts queries for memory fragments from people who use a web browser. It is implemented as a data source as shown in Figure 53. Providing a memory browser over an HTTP connection means that the memory manager is accessible from anywhere on the Internet. In this work, it is used to access the memory manager from a mobile device such

as a PDA as shown in Figure 54. This HTTP server implementation is based on the Cassini.Net HTTP server [ASP.NET, 2005].

Folder based browser

The folder based browser is represented by the graph model shown in Figure 55 and is used for memory retrieval on a desktop computer. The hot folder monitors a folder for files that contain saved queries and pushes them to its sinks. The hot folder query processor extracts queries and sends them to a user query filter which maps them to context descriptors that the memory retrieval data channel can use to retrieve memory fragments. The memory retrieval data channel sends memory fragments back to the hot folder query processor which then outputs retrieved memory fragments to a folder. The folder based browser can also be used on a PDA over a wireless connection by mapping a folder using Resco's file explorer [Resco, 2003] from a desktop computer onto a PDA.

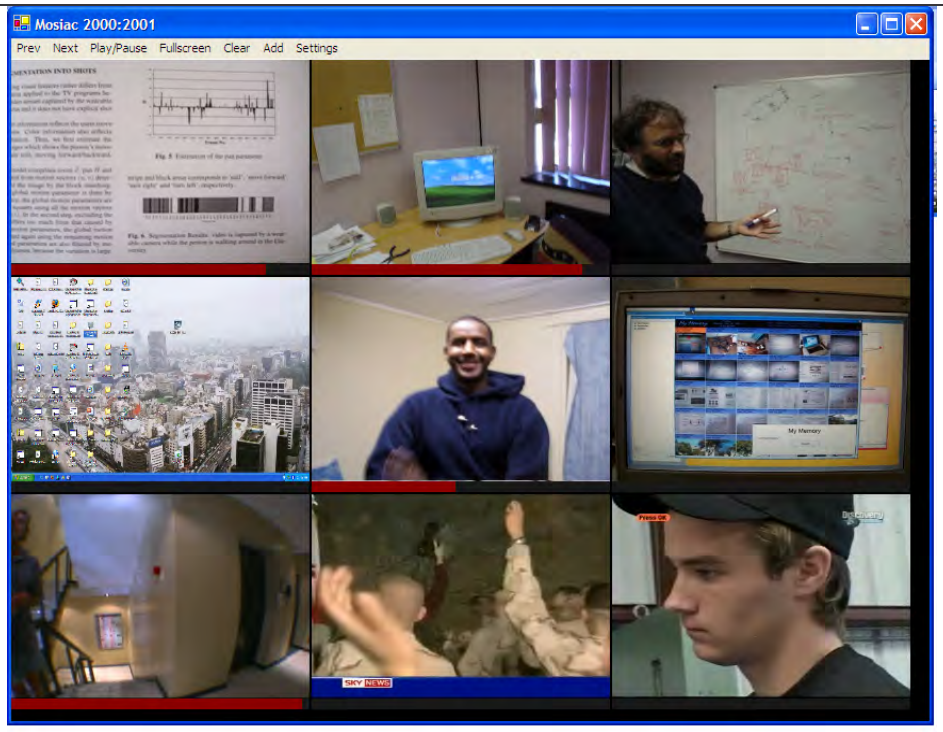


Figure 56. The Mosaic in a 3x3 configuration

The Mosaic

The mosaic memory presentation interface provides a way for consuming multiple channels of visual information at the same time. It can contain *n x m* channels as shown in Figure 56, where *n* is the number of rows of channels and *m* is the number of columns of channels. An individual can direct the results of a query to the mosaic after selecting an existing channel or adding a new one. The mosaic enables users to add a channel, remove a channel, move left, right, up, down or switch to a full screen using the remote control interface provided in section 6.3.4.1. The Nokia 6630 cellular phone used in this work contains arrow keys as well as a central enter button for performing these operations.

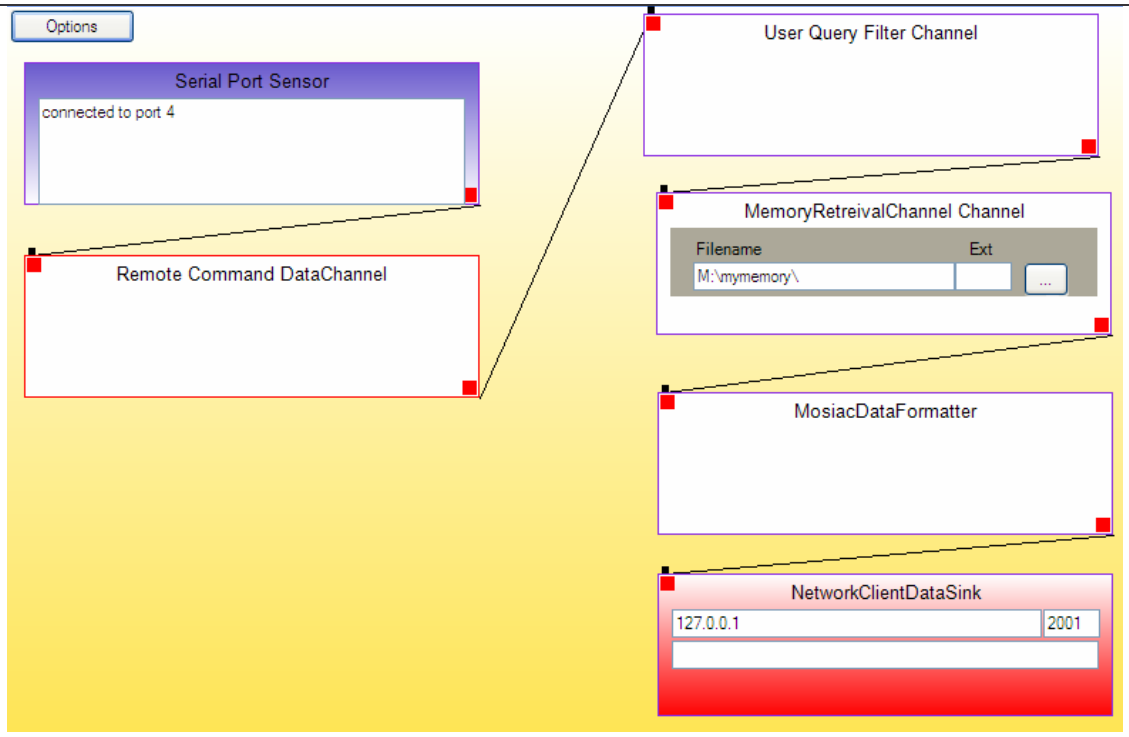


Figure 57. The graph model which represents the Mosaic and remote control based memory retrieval interface

Figure 56 shows the mosaic enabling the consumption of an image of a paper from memory, video of a remote location, past memory of a meeting, contents of a desktop, a communication window to an individual, past memories of interactions with a computer, past interactions with an individual and two TV channels, all at the same time. Each

screen in the mosaic is capable of rendering collections of memory fragments as play lists. Thus individuals can play, pause, play previous and next items in play lists. Each screen in the mosaic is a media player which can render memory fragments. The media player is implemented using Directshow [Netmaster, 2003] and ffdshow [ffdshow, 2005]. The mosaic application is implemented as a server. A network client data sink connects it to a memory retrieval data channel, as shown in Figure 57, through a mosaic data formatter channel which formats retrieved memory fragments for transmission to the mosaic.

The serial port sensor data source, shown in Figure 57, accepts incoming commands from a remote control (cellular phone) and pushes them to the remote command data channel. The remote command data channel maps remote control codes as described in section 6.3.4.1. The user query filter transforms queries to context descriptors. The memory retrieval data channel retrieves memory fragments using the context descriptors. The retrieved memory fragments are then sent to the mosaic through the mosaic data formatter channel.

7 Evaluation of the experimental machine-based memory manager

After developing a model for a machine-based memory manager and implementing an experimental system which is based on the model, it is necessary to evaluate the experimental system. The evaluation serves to establish whether or not the machine-based memory manager fulfils its function, which is *to monitor the states of a person and their environment, capture a person's external memories and manage them with context descriptors in order to enable their retrieval by context*, as specified in the model presented in chapter 4.

In this chapter, a series of tests which are run to evaluate the machine-based memory manager are described. Problems and sources of errors which can influence the outcome of the tests are identified. Experiments during which the tests are run are described. The results which are obtained after running each experiment as well as the conclusions which are reached are presented.

7.1 Tests which make up the evaluation

The evaluation focuses on aspects of the machine-based memory manager which can be tested to establish the fulfilment of its functions as well as its usefulness. Firstly, the ability of its memory manager to accurately store and enable the retrieval of external memories by context. Secondly, its ability to actively monitor, capture and store external memories in order to enable their retrieval by context. Thirdly, through memory retrieval and presentations interfaces, its ability to enable retrieval and presentation of relevant results from the large number of external memories which it manages. Lastly, its performance in terms of use of system resources. This section presents tests which evaluate the above aspects of the machine-based memory manager.

7.1.1 A test of accurate memory fragment storage and retrieval by context

The memory manager uses a context aware file system to manage external memories which it automatically organizes by context as described in chapter 4. To ensure that the memory manager and its context aware file system operate deterministically, it has to be tested in a controlled setting. This test uses a known set of memory fragments and manually supplied context descriptors which are passed to the memory manager for storage. The memory manager is then requested to return memory fragments by context through queries. Prior to the queries being issued, the number and type of memory fragments that each query should return is known. Thus, these values can be compared with those returned by the memory manager after queries to establish whether or not it enables accurate retrieval of memory fragments by context. The experiment in which this test is run is presented in detail in section 7.3.1.

7.1.2 Machine-based memory manager active operation test

The composition of the machine-based memory manager has been described in chapter 5. It consists of a number of subsystems such as video, audio and desktop context descriptor identification and memory fragment capture subsystems. In the active operation test, these subsystems are run and exposed to an environment which contains known context. Context based memory retrieval queries are then performed to determine whether the various subsystems have captured memory fragments and context descriptors accurately. The experiment in which this test is run is presented in detail in section 7.3.2.

7.1.3 Memory retrieval and consumption test using a large data set

As a personal memory manager, the machine-based memory manager is required to keep track of the large number of external memories which accumulate over years of use. Through the various memory retrieval and consumption interfaces which it contains, it should enable the retrieval of memory fragments by context and present relevant results. In this test, a large number of memory fragments are added to the memory manager's context aware file system. A few of the memory fragments are selectively tagged using context descriptors which are accurate (i.e. manually supplied by a human). These

context descriptors are then used to retrieve and consume memory fragments through the machine-based memory manager's various memory retrieval and presentation interfaces. The results are checked manually for relevancy. The results displayed by each memory presentation interface are compared to ensure that they are the same. The experiment in which this test is run is presented in detail in section 7.3.3.

7.1.4 Machine-based memory manager performance test

The performance test measures two aspects of the system. Firstly, the speed at which the memory manager can retrieve memory fragments through context descriptors. Secondly, system resource usage in terms of CPU, memory and thread usage of the various subsystems which make up the machine-based memory manager. Measuring memory retrieval performance is important because it affects the usefulness of the memory manager. A measure of system resource usage is necessary to establish the proper operation of the machine-based memory manager in terms of the software and hardware platform used. The experiment in which this test is run is presented in detail in section 7.3.4.

7.1.5 The evaluation compared to evaluations by other authors

Some of the authors whose works, in machine-based memory aids, are considered in chapter 2 also perform evaluations of their system. These include Aizawa [Aizawa, 2001] who indexes video with brain wave data; Kern [Kern *et al*, 2004] and Vemuri [Vemuri *et al*, 2004] who index audio with text annotation and voice patterns; Hoisko [Hoisko, 2003], Mann [Mann, 2001, 2004], Frigo [Frigo, 2004] and Gemmell [Gemmell *et al*, 2005] who capture memories with their wearable system; and Deutscher [Deutscher *et al*, 2004] who captures memories during social occasions.

The above authors expose their system to the environment to capture external memories during an experiment or use recorded sensor data. Their evaluation occurs over a period of time which varies from hours to days. At the end of an evaluation, descriptions of the kind of external memories their system has captured, and their usefulness, is provided.

Accuracy and usefulness is confirmed by a human, either the person who runs the experiments or participants in an experiment. Aizawa, Kern and Gemmell describe how the additional sensors they used to index video and audio are useful. Gemmell provides measurements of query execution times to show that his database enables fast memory retrieval.

The evaluation which is presented in this chapter combines the above approaches. However, it is different because its aim is not to prove the usefulness of one or two context descriptor sources which are originally developed in this work, as is the case in the above works, where sensor data is used to help retrieve external memories. Instead, it makes use of a variety of context descriptor sources which were originally implemented by other authors. It then shows how multiple context descriptors can be extracted and used by a machine-based memory manager for automatic organization of large numbers of external memories in order to enable their retrieval by context.

7.2 Sources of errors which can affect the tests

Before describing the experiments in which the above tests are run, it is important to consider the sources of errors which can influence the results of the experiments and ultimately the conclusions of the evaluation.

7.2.1 Potential sources of errors

The machine-based memory manager is made up of several components which, according to the model presented in chapter 4, can be categorized as:

1. memory producers (sensors, memory fragment encoders, context interpreters)
2. a memory manager (context aware file system)
3. memory consumers (memory retrieval and consumption interfaces)

The memory manager, memory consumers and the performance of the system as a whole are the subject of the tests described in sections 7.1.1, 7.1.3 and 7.1.4. If any of these

components produce errors, they will be observed in the results of each experiment. Establishing the accuracy of the memory producers used in the test described in section 7.1.2 is more difficult.

7.2.2 Establishing the accuracy of memory producers

Consider a context interpreter which is part of a memory producer. An attempt can be made to measure its accuracy by feeding it large amounts of test data, and using a software component to quantitatively measure its accuracy. This implies that the software component knows how to accurately identify correct and incorrect context interpretation. If such a software component is known to exist, it would be used rather than developing and testing the accuracy of a new context interpreter.

An alternative strategy is to use a human to confirm the accuracy of the context interpreter as humans are very good at interpreting certain contexts (such as those used in this work). This approach is exploited by the authors whose work is compared in chapter 3 (section 3.4.3) and described in Appendix B. One example is Clarkson's location interpreter [Clarkson, 2002]. He recorded video over a 90 day period and also wrote down, on paper, his location when he moved around. He then compared the location identified by his context interpreter against the manually recorded location to establish the accuracy of his interpreter. However, when making use of a human context interpreter, the following problems arise.

Problem 1: The input sample size needs to be small to make it manageable for the individual who is validating correct context interpretation. Otherwise, the possibility for error is high. It can also take a very long time to run the experiment.

Problem 2: Making the input sample size small implies that the probability that the sample is representative of real world data is small, making the evaluation less useful.

The machine-based memory manager developed in this work makes use of more than 30 memory producers. These memory producers are implemented using existing hardware and software libraries. Rigorous testing of each component using sample data which is thought to be representative of real world data is best left to the original implementers of the hardware and software libraries. Instead of concentrating on evaluating the accuracy of each context interpreter, the evaluation relies on the outputs of memory producers whose accuracy can be clearly established.

7.2.2.1 Classification of the accuracy of memory producers

The outputs of memory producers which were observed during the development, testing and debugging of each memory producer used by the machine-based memory manager can be categorized as correct (c), incorrect (ic) and none (n). Partially correct results are put in the incorrect category. (Refer to Appendix C for visual examples which show how memory producers are tested). The list of memory producers and the categorization of their outputs is shown in Table 8. The table also shows whether a memory fragment (mf) or context descriptor (cd) is produced by a memory producer as well as its software and hardware dependency (nc – notebook computer). Based on this categorization, a picture of the accuracy of memory producers can be formed. Memory producers whose output is categorized as “ic” are not always dependable.

It should be noted that, incorrect (ic) production of a context descriptors and their subsequent storage by the memory manager in its context aware file system does not affect the contents of the entire context aware file system (CAFS) or its ability to retrieve memory fragments using other accurate context descriptors. It only affects the usefulness of the single context descriptor as a method of memory retrieval. This is because each context descriptor is stored separately, in the CAFS’s context descriptor layer as described in chapter 4.

	c	ic	n	output	software dependency	hardware dependency
video recorder	x			mf	system API	camera
audio recorder	x			mf	system API	microphone
screen capturer	x			mf	system API	nc
PIM data recorder	x			mf	system API	bluetooth
GPS data recorder	x			mf	system API	GPS device, bluetooth
active application monitor	x	x	x	cd	system API	nc
audio analyzer	x			cd	system API	microphone
email monitor	x		x	cd, mf	third party library	nc
face interpreter	x	x	x	cd, mf	third party library	camera
hot folders	x			cd, mf	system API	nc
keyboard and mouse monitor	x			cd, mf	system API	nc
light level interpreter	x			cd	none	camera
location context interpreter	x		x	cd	system drivers	nc
object recognizer	x	x		cd, mf	third party library	camera
metadata	x		x	cd	third party library	nc
PIM information interpreter	x			cd	system drivers	PIM device, bluetooth
process monitor	x			cd	system API	nc
video motion interpreter	x			cd, mf	none	camera
weblog interpreter	x		x	cd	none	nc
word interpreter	x			cd	third party library	nc
speech	x	x	x	cd	third party library	microphone
OCR	x	x	x	cd	third party library	camera
manual speech based tagger	x	x	x	cd	third party library	microphone
remote control based tagger	x			cd	system drivers + API	PIM device, bluetooth
user query filter	x			cd	system API	nc
manual tagger	x			cd	none	nc

Table 8. Categorization of the outputs of memory producers

7.3 Experiments in which the tests are run

This section presents the experiments in which each of the tests that have been described in the previous section is run. For each experiment, the hypothesis, apparatus, data, experimental procedure, observations, results and conclusions are presented. Two features are common to all the experiments:

1. the apparatus consists of the software and hardware which is described in chapter 6
2. the data consists of context descriptors and memory fragments

7.3.1 Experiment 1: accurate memory fragment storage and retrieval by context

Hypothesis

The memory manager can store memory fragments and enable their retrieval through multiple context descriptors accurately.

Apparatus

Data flow based application modeller

A multiple active hot folder data source

A memory storage data channel

Hard drive volume mounted as m:\

The custom memory browser which is described in chapter 6

The file time changer application which is described in the procedure section of this experiment

Data

40 memory fragments grouped as {a1 to a10}, {b1 to b10}, {c1 to c10}, {d1 to d10}

4 context descriptors cd1, cd2, cd3, cd4

30 memory fragments labelled as {e1 to e12}, {f1 to f9}, {g1 to g6}, {h1 to h3}

4 context descriptors cd5, cd6, cd7, cd8

Memory browser

Memory fragments can be any electronic documents

Context descriptors are the strings “cd1”, “cd2”, …, “cd8”.

Procedure

Preparation of data:

Each group of memory fragments are placed in a directory which represents their context descriptors: directory cd1 - files in group a, cd2-files in group b, …, cd8 - files in group h.

Visually, the distribution of memory fragments on the timeline and the context descriptors which point to them is illustrated in Figure 58.

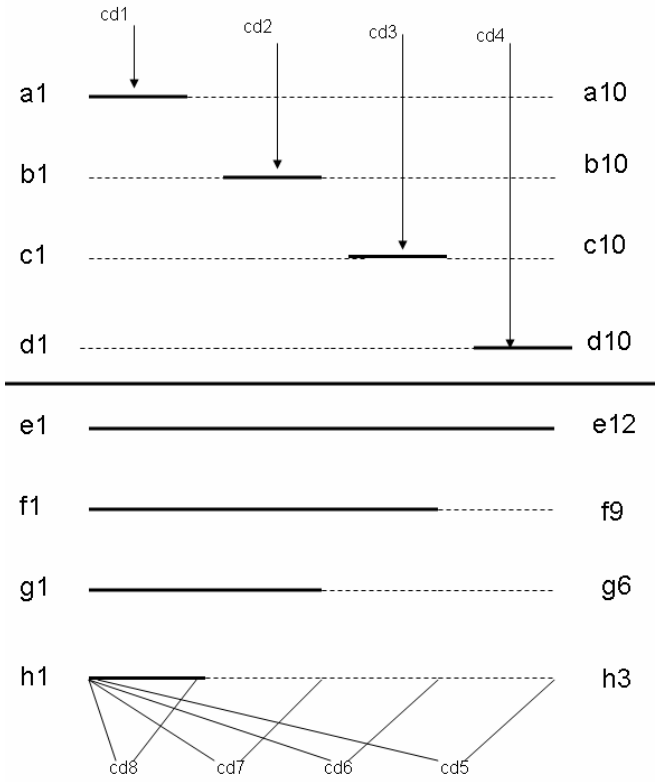


Figure 58. Distribution of memory fragments used in experiment 1 on the time line and the context descriptors which point to them

The last write time and creation time of each group of memory fragments is set as follows.

a1-a10 2005/05/05 12:00:00 to 2005/05/05 13:40:00
b1-b10 2005/05/05 14:00:00 to 2005/05/05 15:40:00
c1-c10 2005/05/05 16:00:00 to 2005/05/05 17:40:00
d1-d10 2005/05/05 18:00:00 to 2005/05/05 19:40:00

The time stamp for a memory fragment in each group differs by 10 minutes
The time period to which each group of memory fragments belongs is different.

e1-e12 2005/05/05 12:00:00 to 2005/05/05 14:00:00
f1-f9 2005/05/05 12:00:00 to 2005/05/05 13:30:00
g1-g6 2005/05/05 12:00:00 to 2005/05/05 13:00:00
h1-h3 2005/05/05 12:00:00 to 2005/05/05 12:30:00

The time stamp for a memory fragment in each group differs by 10 minutes.
The time period to which each group of memory fragments belongs partially overlap.

Modification of the time stamps of memory fragments is performed using the application, called FileTimeChanger (developed during this work), which is shown in Figure 59. FileTimeChanger makes use of the C# FileInfo object to alter the time stamp of a group of files starting from a given time and changing this value incrementally for every file in a group of files. To use it:

- the time stamp of the first memory fragment in a group is set
- the time increment in seconds that will be applied to other memory fragments in a group is set
- a group of files is dragged and dropped on the application's form

The above steps are then repeated for each group of memory fragments.

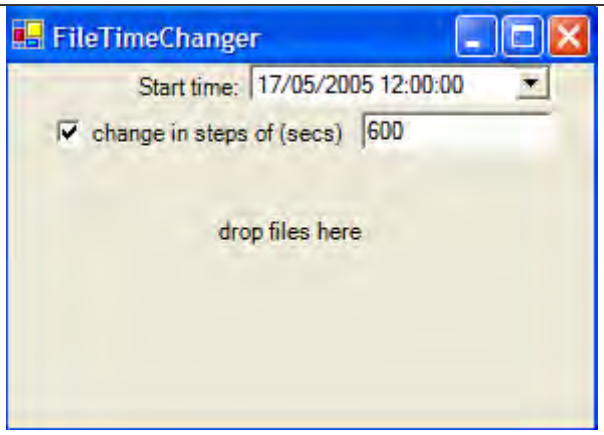


Figure 59. File time changer application

Preparation of apparatus:

Step1: A graph model is created

In the data flow based application modeller (which is described in chapter 4), an empty graph model is created. A multiple active hot folder data source is added to the graph model and put in edit mode. The directories which contain the labelled memory fragments are added to it. The hot folder extracts the name of the directory in which each group of memory fragments is stored and uses it as a context descriptor for the corresponding group of memory fragments. A memory storage data channel is added to the graph model and configured with “m:\memory” as the path to its context aware file system. The hot folder is connected to the memory storage data channel and the graph model is saved as “experiment1.xml”.

Step 2: An empty context aware file system is created.

A 2GB TrueCrypt volume [TrueCrypt, 2005] is created and mounted as “m:\”.

Running the experiment:

The graph model which was created in step1 above is loaded into the graph modeller. This begins the process of addition of memory fragments into the context aware file system. The hot folder detects all memory fragments contained in each of the directories which it is configured to watch. It passes each memory fragment and context descriptor that corresponds to each memory fragment to the memory storage data channel which adds them to its context aware file system. After all memory fragments have been added, the memory browser is run and queries which request for memory fragments by context are executed. The queries which are executed are as follows:

“ipq=100” “mi=100”

{memorytpe:cd1} “ipq=100” “mi=100”

{memorytpe:cd2} “ipq=100” “mi=100”

{memorytpe:cd3} “ipq=100” “mi=100”

{memorytpe:cd4} “ipq=100” “mi=100”

{memorytpe:cd5}{memorytpe:cd6} “ipq=100” “mi=100”

{memorytpe:cd5}{memorytpe:cd6}{memorytpe:cd7} “ipq=100” “mi=100”

{memorytpe:cd5}{memorytpe:cd6}{memorytpe:cd7}{memorytpe:cd8}“ipq=100” “mi=100”

The memory fragment retrieval specifiers “ipq=100” and “mi=100” indicate that at most 100 items per quantum (ipq) and 100 items overall (mi) should be retrieved per query. Since the total number of memory fragments is 70, 100 is a good threshold.

The results returned by the above queries are then checked for accuracy through visual inspection. Each query is repeated 3 times. The entire experiment is repeated 3 times.

Observation

When the experiment is running, the multiple active hot folder data channel shows the number of files it has detected. These decrease as files are added into the memory manager’s storage. The memory storage data channel shows a status bar which indicates its progress in adding memory fragments into the context aware file system. At the end of the experiment, the active multiple hot folder shows that zero files are detected and the memory storage data channel’s progress bar is reset to zero.

Results

For every instance of the experiment which was run and for each query which was executed, the same results are obtained. The results are shown in Table 9.

Query	Retrieved memory fragments	Memory fragment count
“ipq=100” “mi=100”	all memory fragments	70
{memorytpe:cd1} “ipq=100” “mi=100”	a1-a10	10
{memorytpe:cd2} “ipq=100” “mi=100”	b1-b10	10
{memorytpe:cd3} “ipq=100” “mi=100”	c1-c10	10
{memorytpe:cd4} “ipq=100” “mi=100”	d1-d10	10
{memorytpe:cd5}{memorytpe:cd6} “ipq=100” “mi=100”	e1-e9,f1-f9	18
{memorytpe:cd5}{memorytpe:cd6}{memorytpe:cd7} “ipq=100” “mi=100”	e1-e6,f1-f6,g1-g6	18
{memorytpe:cd5}{memorytpe:cd6}{memorytpe:cd7}{memorytpe:cd8} “ipq=100” “mi=100”	e1-e3,f1-f3, g1-g3,h1-h3	12

Table 9. Results of queries run during experiment 1

These results correspond to the memory fragments which each query is expected to return when compared to the illustration of memory fragment distribution previously shown in Figure 58.

Conclusion

The results of this experiment confirm that the hypothesis, which states that the memory manager can store memory fragments and enable their retrieval through multiple context descriptors accurately, is true.

7.3.2 Experiment 2: machine-based memory manager active operation

Hypothesis

In active operation, the machine-based memory manager can monitor, capture a person's external memories, and enable their retrieval through context descriptors.

Apparatus

The wearable computer hardware which is described in chapter 6

A selection of graph models which are described in chapter 6. In this experiment the video, audio, hot folder and desktop graph models are used.

The multiple graph model running application which is described in chapter 5.

A TV set.

Hard drive volumes mounted as c:\ and m:\

The custom memory browser which is described in chapter 6.

Data

Data is automatically acquired from the environment during the experiment. These consist of video frames, audio, desktop computer screens. A news channel on a TV is used as source of video and audio as it is rich in context. The sub-notebook computer (part of the machine-based memory manager) is used as a source of context from a desktop computer. Desktop context is generated by a user which runs various desktop applications.

Procedure

Preparation of data

Data is acquired from the environment during the experiment. A news channel on TV is identified and different desktop applications are identified, e.g. firefox, notepad, winword, pdfreader, vlc, visual studio, thunderbird, explorer, winamp and excel.

Preparation of apparatus

Driver setting of the camera are put in default and automatic mode. The camera is placed ~0.5 meters from a TV set and positioned so that the camera sees the entire TV screen only. The microphone is calibrated using the speech control panel applet which is part of the windows operating system. A new context aware file system is created as described in experiment 1. The sub-notebook computer is put in battery mode. Its maximum performance power setting is selected. Its sleep settings are disabled. Its backlight is set to the lowest settings.

The following directories, which will be used by the machine-based memory manager to store memory fragments, are created:

C:\sensordata\faces, C:\sensordata\capturedscreens, C:\sensordata\keyboarddata, C:\sensordata\liveaudio, C:\sensordata\motionvideo, C:\sensordata\ocrtext, C:\sensordata\speechtext

The corresponding data sinks which are contained in the video, audio and desktop graph models are configured to produce memory fragments into these locations. The multiple passive hot folder data source which is contained in the hot folder graph model (described in chapter 6) is configured to monitor these locations.

Running the experiment

Step1: Acquisition of context from video

The video, audio and hot folder graph models are run using the multiple graph model running application. The experiment is stopped after 3 hours by closing the multiple graph model running application.

Step2: Acquisition of context from desktop

The desktop and hot folder graph models are run using the multiple graph model running application. 10 different desktop applications are run for at least 1 minute per application. The experiment is stopped after 10 minutes by closing the multiple graph model application.

Step3: Validation of the accurate capture of external memories and context descriptors

At this point in the experiment, the context aware file system should contain context descriptors and memory fragments from video, audio and the desktop computer. There is a possibility for running several types of queries to confirm accurate context descriptor and memory fragment capture. These queries include:

Video	Face (one, two, three, multiple)
	ObjectsInVideo (one, two, three, multiple)
	OCRText (word1, word2, ... wordn, indictionary)
	VideoMotion (low, medium, high)
Audio	Noise Type (silent, noise, constantnoise, loudnoise, constantloudnoise)
	SpeechText (word1, word2, ... wordn, indictionary)
	SoundType (speech)
Desktop	ActiveApplication (name)
	AciveApplicationTitle (title)
	KeyboardText (word1, word2, ... wordn, indictionary)
	Keyboard (keydown), Mouse (moved)
	Process (p1,p2,..,pn)
Manual	MemoryType (faces,capturedscreens,keyboarddata,liveaudio,motionvideo, ocrtext, speechtext)

Four of the above context descriptors are selected for use in queries: Face, ObjecsInVideo, OCRText and ActiveApplicationTitle. In each case, the accuracy of the context interpreter used to identify the context descriptor can be clearly established. The face recognizer and object recognizer draw rectangles which match the number of identified faces and objects. Thus, even though these context interpreters may not be accurate, the number of rectangles drawn should match the number of items requested. Similarly, the text identified by the OCR based context interpreter should be contained in captured video frames. The active application title context interpreter uses a Windows system API call which directly reports the name of a window title. Each query is repeated 3 times. The entire experiment is repeated 3 times.

Observation

During the running of the video and audio graph models, the video capture data channel previews the current video frame which is being processed. The hot folder graph model contains a multiple passive hot folder data source which displays the number of memory fragments of each kind that have been captured and passed to the memory manager for storage.

Results

Examples of the results which the memory browser returns after queries are run are shown in Figure 60. In each case the results are relevant (i.e. the retrieved memories correctly correspond to context descriptor based queries). The queries which correspond to the results shown in Figure 60 are:

{face:one} "md=1" images "ipq=2"

asks for at most 2 images per quantum that were captured when the face recognizer detected one face. A matching distance of 1 second is also specified.

{objectinvideo:three} "md=1" images "ipq=1"

asks for at most 1 image per quantum that was captured when three objects where detected by the object recognizer.

{ocrtext:news} "md=1"

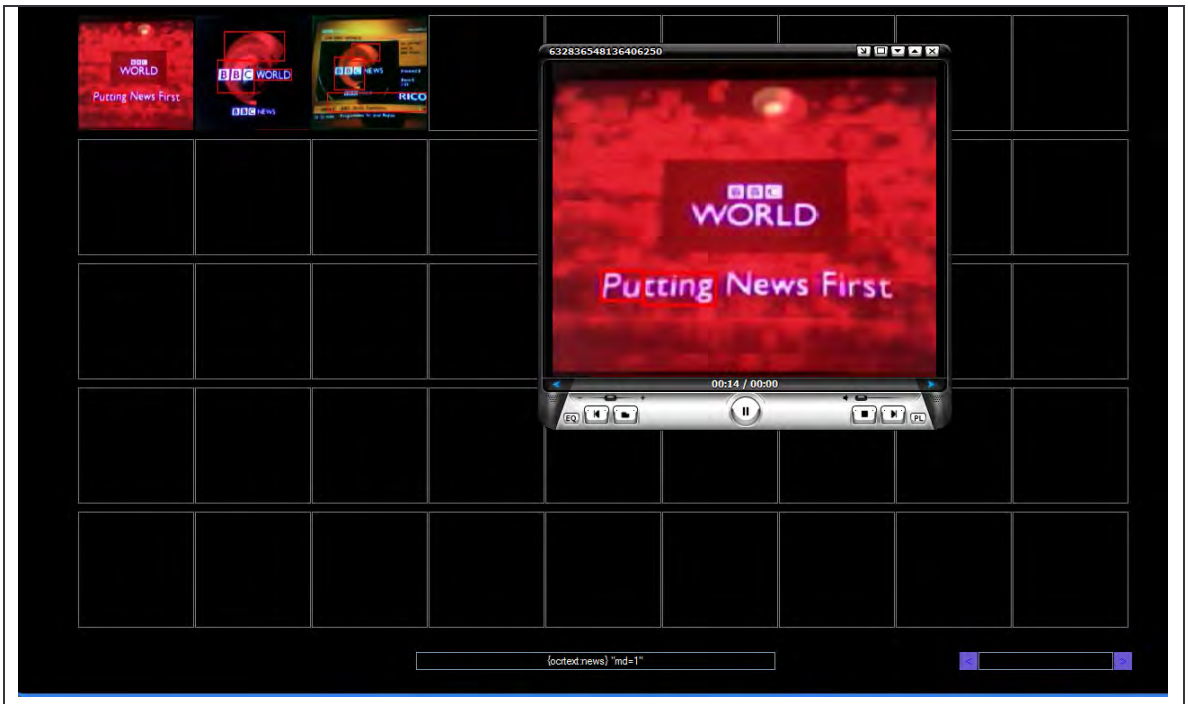
asks for memory fragments that were captured when the text "news" was detected by the OCR based context interpreter.

{activeapplicationtitle:firefox} images "ipq=20" "md=1"

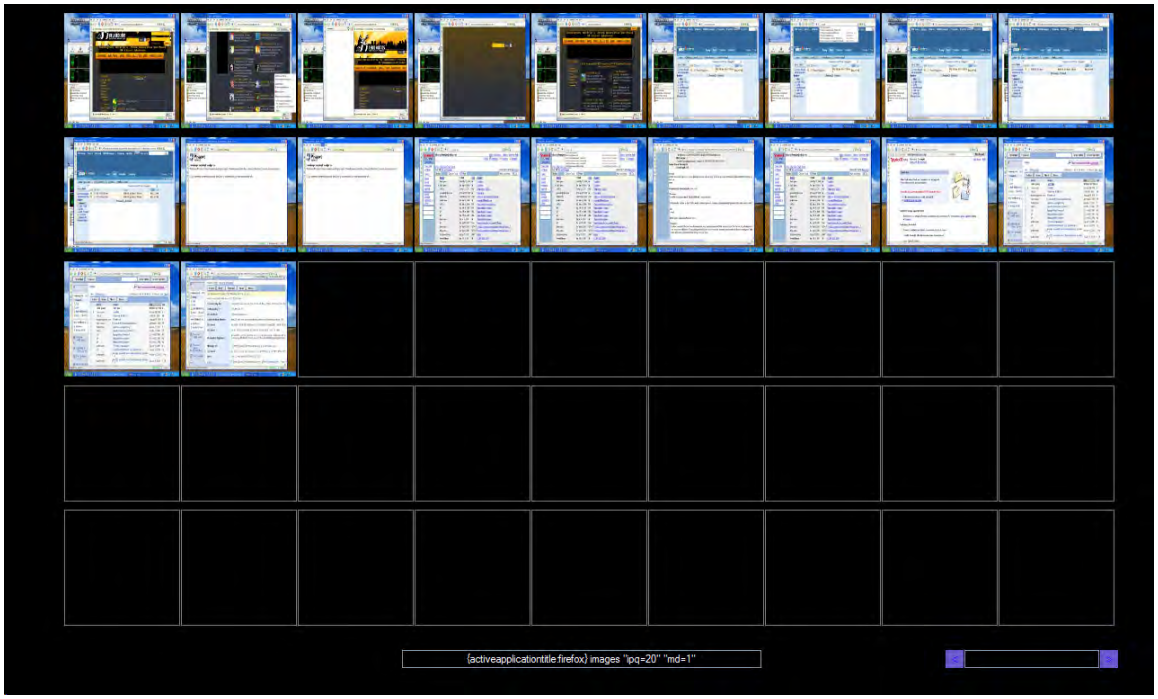
asks for at most 20 images per quantum that where captured when the application firefox was running.

Query: return images which were captured when the face recognizer detected one face

Query: return images which were captured when the object recognizer detected three objects



Query: return memory fragments which were captured when the OCR interpreter detected the text "news"



Query: return images which were captured when the active application was firefox

Figure 60. Results returned by the memory manager in response to context descriptor based queries

Conclusion

The results of this experiment show that the machine-based memory manager can monitor and capture some of a person's external memories and enable their retrieval, through context descriptors.

7.3.3 Experiment 3: memory retrieval and consumption from a large data set

Hypothesis

The machine-based memory manager can manage large numbers of different types of memory fragments and enable retrieval and presentation of relevant memory fragments through its various memory retrieval and presentation interfaces.

Apparatus

Same apparatus as for experiment 1 as well as the memory retrieval and consumption interfaces which are described in chapter 6

The HTTP based memory retrieval data channel

The Mosaic

Remote control (Nokia 6630 cellular phone)

Remote control based memory retrieval graph model which is described in chapter 6.

A web browser

Data

A collection of personal memory fragments accumulated over a three year period consisting of:

~1200 pictures and videos taken with a 5 mega pixel digital camera

~1100 pictures and videos taken with a cellular phone camera

~100 videos recorded with wearable computer at various locations and activities such as office, shopping centre, street, inside car, meetings

~3000 papers (pdf, word, html and text documents) collected over a 3 year period

~120000 electronic documents (html, exe, zip files) downloaded daily from the internet over a 3 year period, 90 % of which are html documents.

~500 email messages spanning a three year period

~300 phone call audio recordings and associated phone numbers

~50 calendar and contact PIM messages sent from cell phone to the wearable computer

Procedure

Preparation of data

All items are stored in a directory per respective memory fragment type.

Preparation of apparatus

A similar graph model to that used in experiment 1 is created. The hot folder used is configured using the directories which contain the memories described in the data section. The graph model is saved as experiment3.xml. An empty context aware file system is created as described in experiment 1.

Running the experiment

The graph model described above is loaded. This starts the process of addition of memory fragments into the context aware file system. The hot folder automatically detects all memory fragments and manually supplied context descriptors and passes them to the memory storage data channel for storage. Due to the large number of memory fragments involved this step of the experiment takes several hours to complete.

After the experiment is compete, the custom memory browser is used to select a few groups of memory fragments and tag them with manually supplied context descriptors. In this experiment, images of a laptop computer, images and videos taken during a meeting, and images taken while dining are tagged.

Queries are then executed to retrieve the tagged memories using the memory browser, the HTTP based memory browser and the mosaic. The results are manually checked for correctness (should match those tagged manually).

The queries consist of:

{object:laptop}
asks for memory fragments which represent a laptop computer

{location: coe} images videos "ipq=2"
asks for memory fragments formed in the COE building (CS department building)

{profile:meeting}{location: coe} images videos "ipq=2"
asks for memory fragments formed in the COE building during meetings

{profile:meeting} {location:coe} {person:shaun}{person:alf} images videos "ipq=2"
asks for memory fragments which were created during a meeting at the COE when both Shaun and Alf were present.

{profile:dining}
asks for memory fragments which were created during a dining profile

For use with the remote control the following entries are added:

300#| {object:laptop}
301#| {profile:meeting} images videos "ipq=2"
302#| {profile:dining}

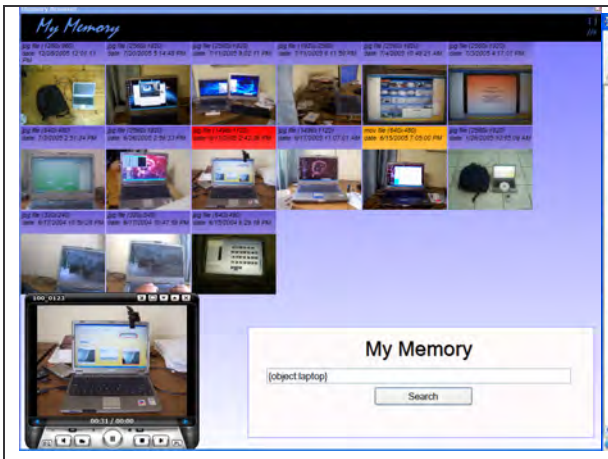
to the remote control map file. The remote control based retrieval graph model is loaded. The above key codes are executed to run the corresponding query. The results are checked manually for correctness as the Mosaic presents a slideshow.

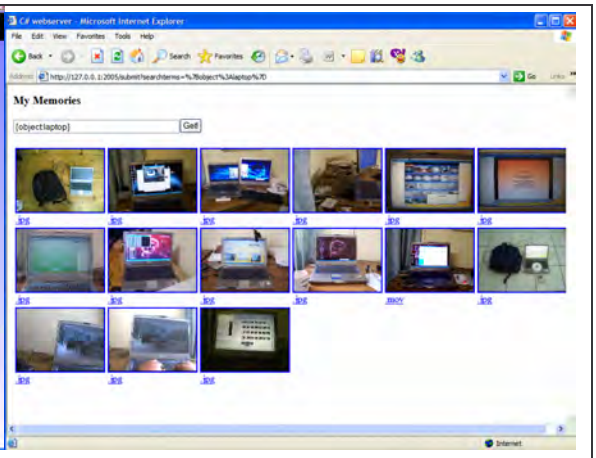
Observation
The memory storage data channel shows a status bar which indicates its progress of adding of memory fragments into the context aware file system. At the end of the

memory fragment addition process, the active multiple hot folder shows that zero files are detected and the memory storage data channel's progress bar is reset to zero.

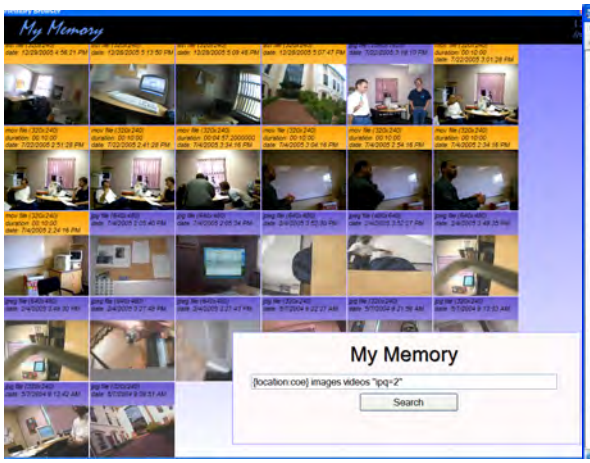
Results

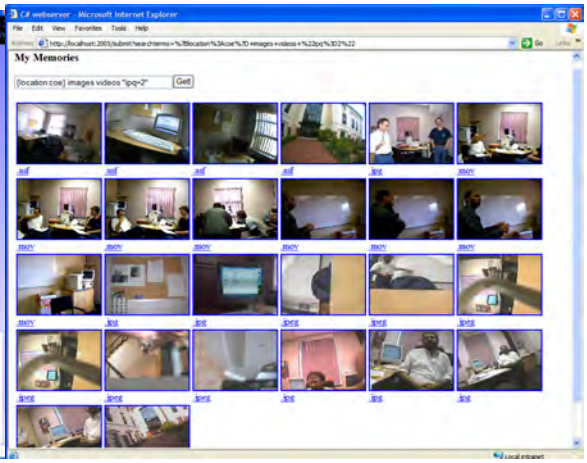
The results of running retrieval queries which are targeted at selectively tagged memory fragments are shown in Figure 61. The results show that even though there are a large number of memory fragments being managed, the memory manager enables accurate selective retrieval of memory fragments by context. The two different memory retrieval interfaces (memory browser and web browser) also display the same results.



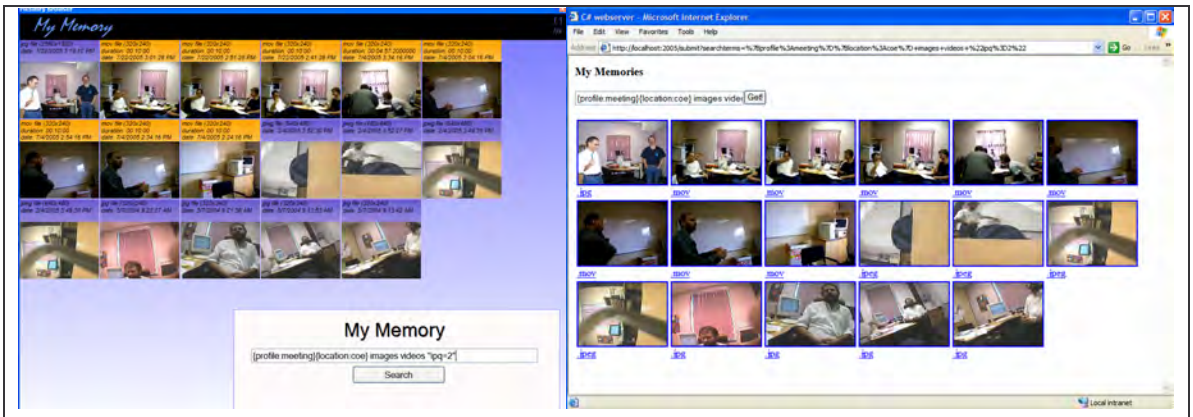


Query: return memory fragments of a laptop computer

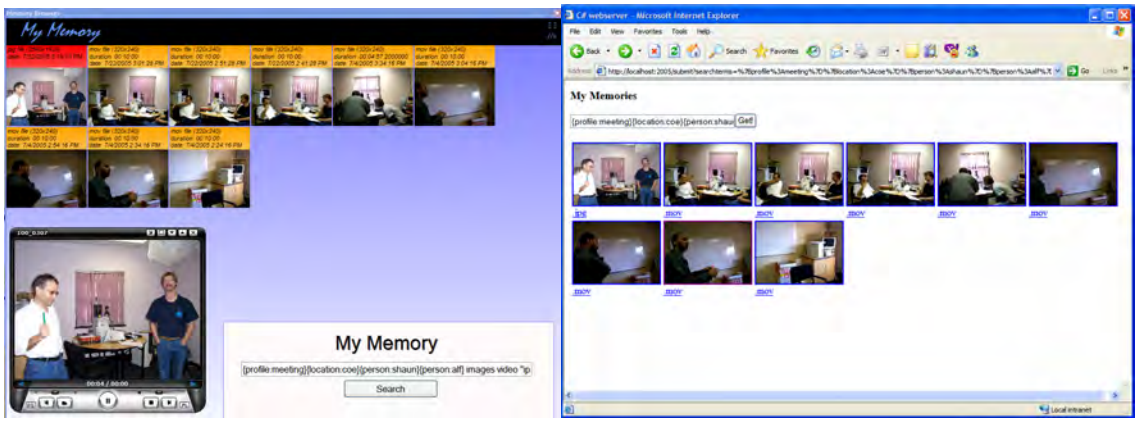




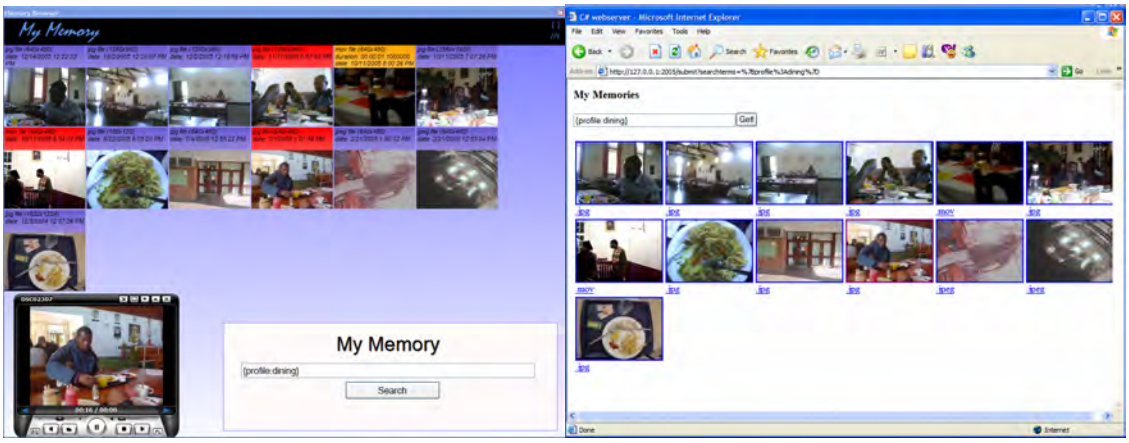
Query: return memory fragments formed in the COE building



Query: return memory fragments formed in the COE building during a meeting



Query: return memory fragments formed in the COE building during a meeting when both Shaun and Alf were present



Query: return memory fragments captured while dining

Figure 61. The memory browser presenting the results of various queries

While the two interfaces shown in Figure 61 enable the consumption of results of single queries, the Mosaic, shown in Figure 62, enables the consumption of the results of multiple queries at the same time.

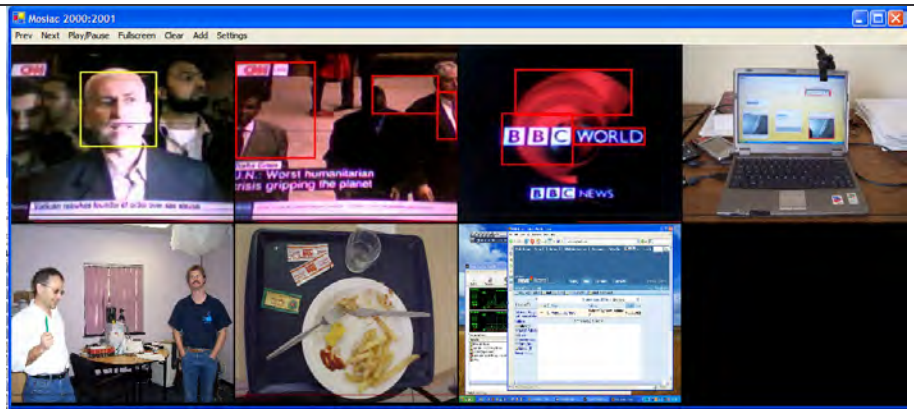


Figure 62. The Mosaic showing the results of queries executed during experiment 2 and 3 at the same time

Conclusion

The results of this experiment support the hypothesis which states that the machine-based memory manager can manage a large number of personal memories and enable the retrieval and presentation of relevant memory fragments through its memory retrieval interfaces by context.

7.3.4 Experiment 4: memory manager performance

Hypothesis

Hypothesis 1: Context descriptor based memory retrieval provided by the memory manager is reasonably fast (takes less than 5 seconds).

Hypothesis 2: The memory manager makes proper use of system resources such as CPU, memory and threads.

Apparatus

An application which can execute a given number of queries one after another a second apart (to simulate the issuing of queries by a user), called benchmarkRetrieval.exe

The various graph models described in chapter 6 which are part of the machine-based memory manager (video, audio, desktop, PIMGPS, hotfolder and memory retrieval graph models)

Multiple graph model running application.

Data

Context aware file system which contains the memory fragments from experiments 2 and 3.

Procedure

Preparation of data

Experiments 2 and 3 are run to add memory fragments and context descriptors into the context aware file system.

Preparation of apparatus

The machine-based memory manager is put in debug mode by creating the file "debugmode.txt" in the software environments application directory. In debug mode, the machine-based memory manager keeps statistics for every memory retrieval such as the time taken for stage 1 (mapping context descriptors to quanta), stage 2 (retrieval of memory fragments from quanta) and overall memory retrieval time. benchmarkRetreival.exe outputs this data to a log file called "log.txt" when given a file called "queries.txt" containing multiple queries. The memory manager also keeps track of CPU, memory and thread usage of each instance of a graph model which is currently running. This data is recorded into a time stamped file which is stored in the logs directory under the software environment's application directory.

Running the experiment

Step1: query performance measurement

benchmarkRetreival.exe is run with an input file which contains 50 context descriptor based queries similar to those presented in experiments 1, 2 and 3. The outputs of benchmarkRetrieval are imported into excel and graphed.

Step 2: subsystem performance measurement

Each individual graph model is run using the multiple graph model running application for 10 minutes each. All the graph models are also run at the same time. The log files

which contain performance data of each graph model are imported into excel and graphs which represent the data drawn.

Observation

When all the graph models are run at the same time, the sub-notebook computer runs noticeably slowly. This interferes in the running of other desktop applications. However, running individual graph models does not have the same effect.

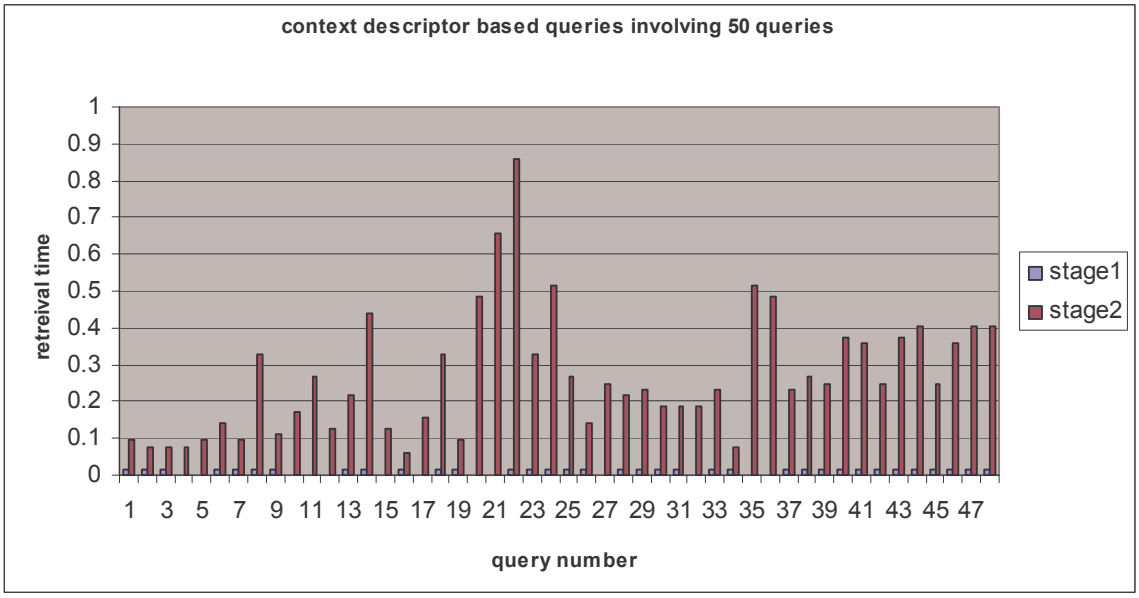
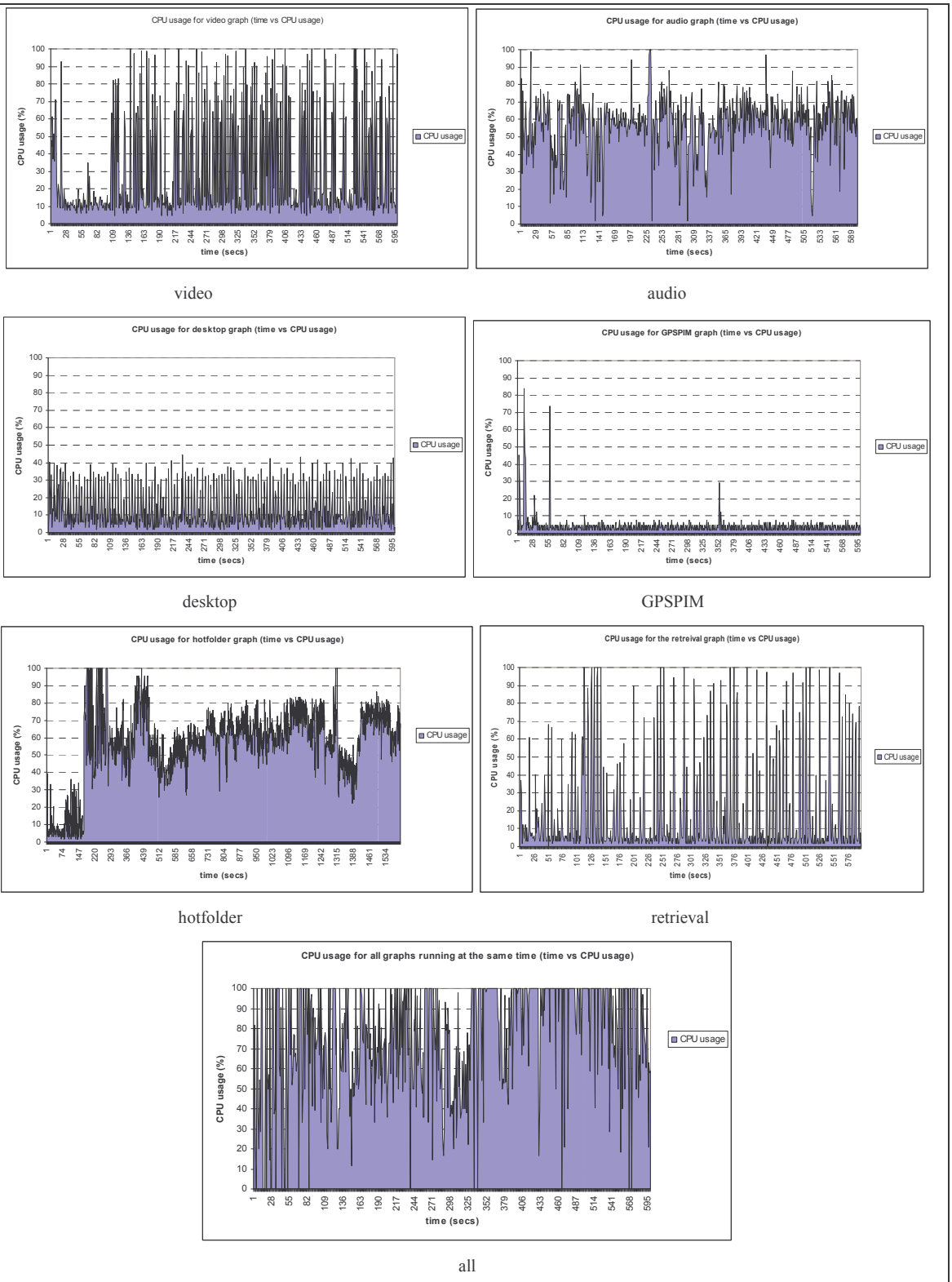


Figure 63. Retrieval performance for 50 context descriptor based queries

Results

Memory retrieval times for 50 queries are shown in Figure 63. On average, context descriptor based memory retrieval takes 0.15 seconds. Stage two of memory retrieval (retrieval of memory fragments from quanta) is slow because it involves fetching memory fragments from secondary memory.



video

audio

desktop

GPSPIM

hotfolder

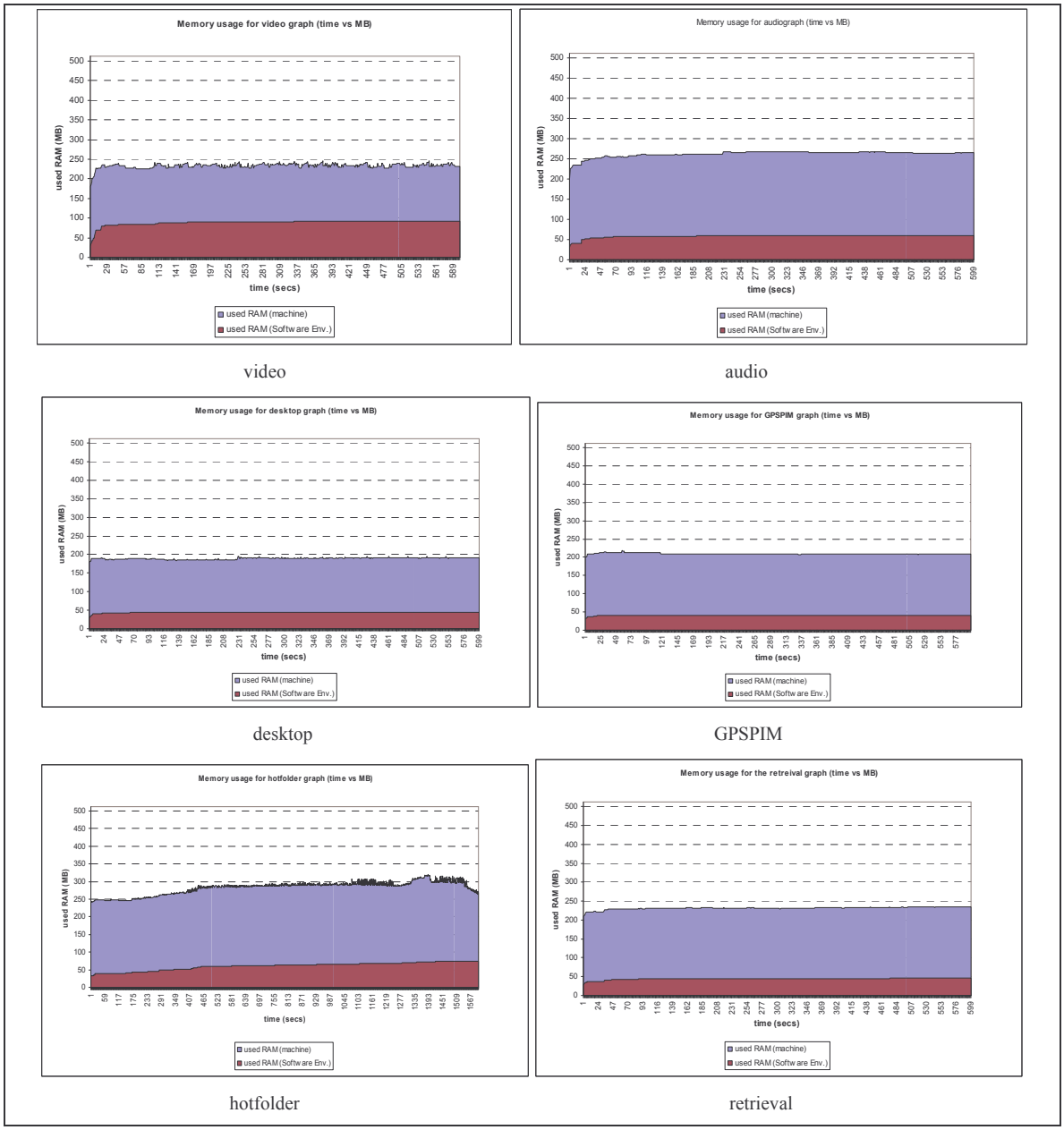
retrieval

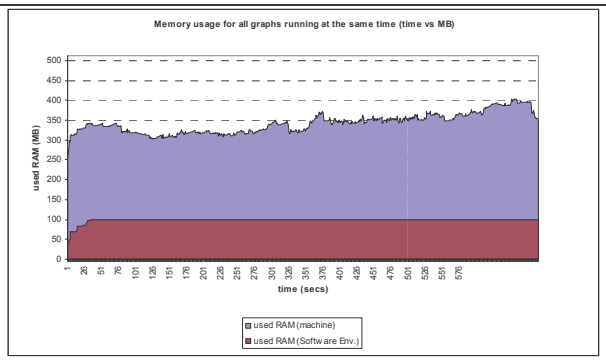
all

Figure 64. CPU usage for graph models used by the machine-based memory manager

CPU usage

CPU usage by each of the graph models that make up the graph models is shown in Figure 64. The speech based context descriptor and hot folder graph models consume the largest amount of CPU resources. In the case of the speech recognizer, the SAPI speech recognition engine consumes most of the resources. In the case of hot folders, the graph shows active indexing of thousands of memory fragments in a short period of time, which is not a normal occurrence in everyday use of the machine-based memory manager.





all

Figure 65. Memory usage for the graph models used by the machine-based memory manager

Memory usage

Memory usage by the machine-based memory manager over a 10 minute period for each graph model is shown in Figure 65. The blue area shows total memory use by the machine, including the operating system, whereas the purple area shows memory use by the graph models. Memory usage is constant for all graphs which indicates that there are no significant memory leaks in the system. When running all the graph models at the same time at most 400 out of 512MB of RAM is used by the machine-based memory manager.

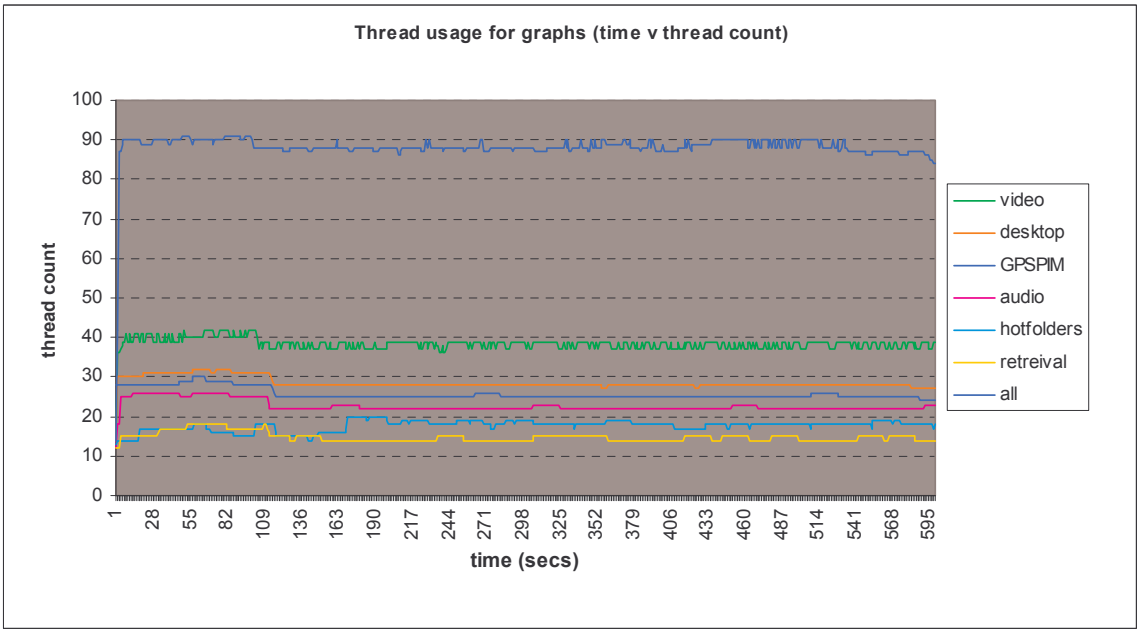


Figure 66. Thread usage for all the graph models used by the machine-based memory manager

Thread usage

The various subsystems of the machine-based memory manager use multiple threads as shown by the graph displayed in Figure 66 which represents thread usage by each graph model over a 10 minute period. There is at least one thread used in every black box (data source, data sink and data channel) as well as threads used for managing queues that connect black boxes. The figure shows that thread usage is constant which means that the graph based simulation and application modeller properly manages multiple threads.

Conclusion

The results of the experiment support the hypothesis which states that

- the memory manager provides reasonably fast memory retrieval (<5 seconds).
- the memory manager makes proper use of system resources such as CPU, memory and threads

7.4 Conclusion

The results of the experiments which are presented in the previous section show that the machine-based memory manager developed during this work (and presented in chapter 6) operates as described by the model presented in chapter 4. It is able to capture an individual's external memories and identify multiple context descriptors which are used by its context aware file system to keep track of them. It is able to absorb large amounts of personal memories which accumulate over a long period of time and enable their retrieval by context. The memories it retrieves are relevant (they match context descriptor based queries) and they contain more detail than are remembered by an individual.

The significance of the results of the evaluation is that, given the state of current technology, it is possible to create a machine which stays in the background and captures day-to-day external memories and organizes them by context. This enables an individual to delegate the responsibility of remembering specific details about his day-to-day existence to a machine and put more focus on accomplishing the current task at hand.

8 Conclusion and future work

At the beginning of this thesis the motivation for a machine-based personal memory manager which can support and augment an individual's biological memory was presented. Subsequently, a model for such a memory manager was developed and used to create an experimental machine-based personal memory manager which can keep track of an individual's day-to-day, external memories. The machine-based memory manager was evaluated and shown to be useful in supporting and augmenting an individual's biological memory system.

In this chapter, an overview of the thesis which shows how the development of such a machine-based memory manager was achieved, is presented. The contributions made by this work are identified. A critical analysis of the contributions is presented. The challenges faced during this work are considered. Future work which is necessary in order to improve the machine-based memory manager is identified.

8.1 Overview of the thesis

Chapter 2 established that people have personal memory management problems and that they need a machine-based personal memory manager to help them keep track of their external memories. It showed that current personal memory management software and hardware is inadequate for managing an individual's day-to-day memories while proposals for future machine-based memory aids are promising. Consideration of psychological models of memory provided one possible interpretation of how the human memory system works. Consideration of clinical psychology works showed the weaknesses which are inherent in the human memory system, such as forgetting, which is identified as a fundamental weakness, but can also be beneficial. Consideration of proposals for future machine-based memory aids showed that although researchers are aware of the need for a machine-based memory manager that supports and augments an individual's biological memory and can deal with memories which accumulate over long periods of time, no comprehensive solution to the problem had been proposed.

Chapter 3 identified context aware computing as a solution for day-to-day personal memory management. After considering existing definitions of context it presented a precise definition of context which is relevant to machine-based memory management. Consideration of existing context based memory aids showed that proposed systems were limited by the variety of context descriptors they used to manage an individual's external memories. The most often used context descriptors were location and manually supplied text annotation. There was no proposal on how data which accumulates over long periods of time can be managed using a variety of context descriptor sources and how these context descriptor sources should be managed. In context aware computing, a number of architectures which could be used for developing context aware applications are proposed. However, these architectures were found to be less useful for direct use in developing a machine-based memory manager because they are interested in providing a multitude of applications rather than focusing on context based memory management (i.e. they are application centric rather than memory centric). As a result the need for a memory centric context model was identified. Consideration of existing context aware application development approaches showed that implementing context aware architectures for experimentation using existing system development tools is complex. Such an undertaking requires a lot of time and resources. Developing a framework and a dedicated software tool that can be used to prototype context aware applications which reduces development time, complexity and reliance on availability of resources, such as sensing hardware, was identified as important.

Chapter 4 developed a model for a machine-based personal memory manager which adapted the information processing model of memory and context based memory management. It explained machine-based memory management in simple terms using a number of entities: lifestreams, memory producers, a memory manager, memory consumers, context descriptors and memory fragments. It introduced a context aware file system which is used for context descriptor based lifelong memory management. The model succeeds in describing external memory management by a machine from the time memories are perceived by a machine to the time they are requested and used by an

individual. In chapter 2, the need for such a model is identified. Therefore the model is one of the contributions of this thesis.

Chapter 5 conceptualized and presented the implementation of a software environment which can be used to prototype context aware systems. The necessity of such a system had been established in chapter 3. Context aware systems make use of multiple sensors, software components which analyze sensor data as well as components which present information to people. Building such systems is a complex undertaking. The software environment was therefore created to enable individuals to build context aware systems with the least amount of effort. It represents a single environment that contains all the facilities needed to build context aware applications. In the software environment, large systems are built using small black boxes through a visual programming interface. In chapter 3, the need for the software environment is identified. Therefore the software environment is one of the contributions of this thesis.

Chapter 6 described the implementation of an experimental machine-based memory manager which is based on the model proposed in chapter 4. To construct the hardware components of the machine-based memory manager, hardware consisting of widely available components such as a general purpose computer and multiple sensors were used. To construct the software components of the machine-based memory manager, existing as well as new implementations of sensor drivers, context interpreters and external memory presentation interfaces were used. The software components were implemented in the form of black boxes for use in the software environment's visual programming environment which simplified the implementation of the machine-based memory manager. The experimental machine-based memory manager is identified as one of the contributions of this thesis. It represents an instance of a context aware machine which monitors an individual and his environment, captures his external memories and enables their retrieval with multiple context descriptors.

Chapter 7 presented an evaluation of the machine-based memory manager. The evaluation consisted of experiments during which the ability of the machine-based memory manager to enable context based external memory management was tested. The results of the experiments showed that (1) the machine-based memory manager is able to accurately store and enable the retrieval of external memories through multiple context descriptors, (2) in active operation the machine-based memory manager is able to capture and manage external memories by context and (3) it is able to manage large numbers of external memories and enable retrieval of relevant external memories by context. The conclusion reached at the end of the evaluation is that the machine-based memory manager is able to manage large amounts of personal memories and enable their retrieval by context. Therefore, it is able to support and augment an individual's biological memory system. This, in turn, implies that the model on which the machine-based memory manager is based is useful for constructing a machine-based memory manager but does not imply that the model is valid.

8.2 Contributions

The main contribution made by this thesis is the development of a machine-based personal memory manager. The machine-based memory manager is a ubiquitous application which has the potential to affect the life of an individual and the way he makes use of applications and devices. The intended effect of the machine-based memory manager which is developed during this work is to support and augment an individual's biological memory. The main contributions can be divided into the following categories.

Development of a model for a machine-based memory manager

The model describes what a machine-based memory manager is. It explains what a machine-based memory manager should consist of. It explains how a machine-based memory manager should manage an individual's personal memories which accumulate over his lifetime. The model serves as a foundation which individuals can utilize to build a machine-based personal memory manager.

Conceptualization and implementation of a software environment

The software environment brings visual programming to context aware as well as ubiquitous application development. It represents a software infrastructure which enables simplified and rapid construction of complex systems. It provides facilities which encourage less dependence on hardware when prototyping applications.

Implementation of an experimental machine-based memory manager

The implementation of the experimental machine-based personal memory manager serves as a proof of concept. It represents a working example of a machine-based personal memory manager which is built using the software environment that is developed during this work.

8.3 Critical analysis of contributions

The contributions of the thesis address each of the problems which are identified in chapter 1. The main problem which is addressed is personal memory management. The proposed solution to this problem is the development of a machine-based memory manager which takes on the responsibility of keeping track of an individual's external memories. This section discusses each of the contributions of the thesis and their role in the development of the machine-based memory manager. At the same time, it identifies and addresses aspects of the contributions which may be perceived as weaknesses.

The model for a machine-based memory manager

The model combines two existing models of memory, the information processing model of memory from cognitive psychology and context based memory management from context aware computing. The information processing model of memory has been around for more than 50 years and context aware computing has gained recognition in the last 15 years. The abstractions which are provided by these models are used to conceptualize and develop essential components of a machine-based memory manager and explain how the machine-based memory manager can perform its function of external memory management.

The information processing model of memory has been criticized for being too simplistic in describing how the human memory system works. For example, it does not consider people's psychological and physiological modes of operation. However, as the model which is proposed in this thesis describes the operation of a machine rather than a biological entity the information processing model is suitable. Furthermore, its weaknesses can be addressed. For example, the model proposed in this thesis makes provision for the capture of the psychological and physiological states of people through memory producers and the provision of psychological and physiological modes of operation through memory consumers.

The model of memory management which has been adapted from context aware computing and is used by the memory manager can be criticized for utilizing the symbolic processing approach for memory management, rather than a connectionist approach which is based on biological systems of memory such as artificial neural networks. As managing an individual's external memories requires a machine to be good at solving perceptual problems, it would seem that a model for a machine-based memory manager should also be based on a connectionist approach. However, this thesis believes that producing a model of memory management which is based on biological models is contradictory to the initial motivation of creating a machine-based memory manager which is to overcome the weaknesses inherent in a biological memory system (the human memory system). As a result, the model provides a memory manager which uses a symbolic processing approach for memory management and does not have weaknesses such as forgetting. The connectionist approach for solving perceptual problems is not abandoned by the model. Such an approach can be used by memory producers for identification of context descriptors.

Development of the concept and implementation of a software environment for prototyping context aware applications

System development using the model of programming which is provided by the software environment (visual and black box based programming) is not a new concept. However, the single complex system development environment that the software environment

represents is a new concept. The alternative to the software environment is to make use of multiple, self contained system development tools. The problem with such an approach is that in addition to having to deal with solving a particular problem, an individual is exposed to the complexity which is inherent in dealing with multiple independently developed system development tools and libraries whose interfaces have to be studied. In contrast, the software environment contains collections of black boxes whose function and interfaces are well known. These components can be assembled quickly using a visual programming interface to form new systems which are easier to maintain as well as extend in the future. The software environment is a complex system. However, it succeeds in hiding its complexity from individuals that make use of it.

Implementation of a context aware machine-based memory manager

The machine-based memory manager implemented in this work manages an individual's external memories by context. However, it does not manage every possible type of memory fragment using every possible type of context descriptors which are part of a person's life. Therefore it can be seen as a partial implementation of the model proposed in this work. This is as a result of the number of sensors and context interpreters which are used by the implementation. Sensor and context interpreter use is limited by the availability of these components. Rather than serve as a full implementation of the machine-based memory manager described by the model developed in this thesis, this implementation serves as an example which uses multiple sensors and context interpreters and can easily be extended in the future. The implementation is a working example of a machine-based memory manager which is entirely dependent on sensors and context interpreters. If the components used by the machine-based memory manager are replaced with new, more reliable and high quality components, the reliability and usefulness of the machine-based memory manager will increase.

8.4 Challenges faced

A number of challenges were faced during this work. The first challenge was addressing the issues of personal memory management. Initially, an attempt was made to manually create a model of an individual's life and use this model to identify memories which could to be captured and managed by a machine. Such a model would be used to identify events and repetitive patterns in an individual's behaviour over a period of time. The identified patterns would then be used to keep track of memories. Later, it became apparent that a manually built model would be an over-simplification of reality and be of little use in managing an individual's life long memories. Moreover, such a model would not be able to predict changes in the behaviour of an individual which can occur in the future. The approach which was adopted is one that the human memory system uses. Humans build a model of their world from childhood through experience (an approach which has consensus amongst cognitive psychologists). Following this well established approach, the concept for a context aware file system (CAFS) was conceived. The CAFS is not directly comparable to the model of memory management used by the human memory system but the basic method which is used for memory management is similar: *memories are managed using an automatically built model of the world which is dependent on an individual's experience*. The CAFS stores memories organized by context as they are captured by the machine based personal memory manager, thus a context based model of an individual's personal memories is built, automatically, from his experience.

The next challenge faced was to fully conceptualize the context aware file system (CAFS). The final version of the CAFS which is described in chapter 3 was developed iteratively. Firstly, the quanta file system which enables the storage of life long memories arranged by their time of formation was conceptualized. Secondly, the concept of storing context descriptors alongside memories in quanta was conceptualised. However, in this configuration, context descriptors could only point to memories which are contained inside a quantum. Lastly, the concept of a context descriptor layer was conceptualised to enable global (inter quantum) association of memory fragments. Due to these

characteristics, context descriptor based memory retrieval from the CAFS is fast (less that 1 second) and independent of the number of memory fragments which are stored in it. It also returns memory fragments which are contextually relevant. The initial implementation of CAFS used 8 byte time stamps (quantum pointers) per context descriptor state in the context descriptor layer. Later, entries in the context descriptor layer were converted into bit arrays, which meant that 1 bit rather than 8 byte quantum pointers were used. This reduced the storage space required for entries in the context descriptor layer by 64 times. For a life long memory storage system, such a saving is significant. The quanta file system, in addition to storing actual memory fragments, also stores links to memory fragments which means that the CAFS is not limited by the size of its internal storage system. It can, for example, store links to memory fragments which reside on the Internet where storage space is plentiful.

The third challenge faced was to develop a working prototype for a machine-based memory manager which is based on the model (presented in chapter 4) in order to demonstrate the practicality of the model. Because such a prototype consists of multiple components, the implementation of the prototype provided a lot of technical challenges. Examination of proposed architectures and methods of building context aware systems led to the development of the concept of a software environment for prototyping context aware applications. The software environment reduced the technical challenges which are inherent in developing a machine-based memory manager. It provided a means of managing the multiple components which make up the machine based memory manager visually. It reduced the act of complex system building into connecting multiple black boxes through a visual interface.

The fourth challenge faced was to identify suitable existing hardware and software solutions that could be integrated into the software environment to create components that could be used to build a context aware machine-based memory manager. In addition, work was necessary to integrate solutions which are developed by a number of individuals and institutions, each of which uses their own interface and implementation style. The software environment’s black box architecture helped in this respect, as each

existing solution could be packaged as a black box and tested individually. In some cases, components such as sensor drivers and context interpreters were implemented in this work from scratch because existing equivalents were not found. Having observed that existing proposals for context based memory management systems made use of very few sets of sensors and context interpreters, the identification of and use of multiple sources for context descriptors as well as sensing hardware was set as a goal. Because it was known that limitations of existing sensing hardware and context interpreters could not be overcome, a manual method of obtaining context descriptors from individuals was also devised and used.

The last challenge faced was to design suitable experiments which could be used to evaluate the machine based personal memory manager. Due to the variety of subsystems it contains, the number of experiments which could be described is large. However, in chapter 7, a few experiments which demonstrate and evaluate the basic principle of context descriptor based memory management were presented. The experiments are designed and documented in order to allow others to be able to repeat them. The use of the software environment helps in this regard by providing an experimental platform which any individual can use without worrying about low level implementation details.

8.5 Future work

A number of possibilities exist for future research. The machine-based memory manager needs more automatic memory producers. The provision of services to individuals such as event notification, personal time planning and oracles [Chavez *et al*, 1999] using the memory centric model which is developed in this thesis needs investigation. Extension of the visual programming interface of the software environment to enable other modalities of programming such as speech and thought based programming needs investigation. Various experiments which test the usefulness of the machine based memory manager in a real world setting need to be designed and run. An example scenario is in testing its usefulness in augmenting the memory of old people.

With respect to the implementation, the software environment developed in this work should be moved to a more open platform such as Linux instead of being dependent on the proprietary Windows operating system which is currently used. Compilers and common language runtime environments (CLRs) which allow C# code compilation and execution on open platforms such as mono.NET [mono, 2005] and DotGNU [GNU, 2005b] are improving. In the future, they will be able to compete with Microsoft's implementation. As hardware technology advances, it should be possible to assemble a more transparent personal memory manager in a smaller form factor [LinuxDevices, 2005], containing sensors such as tiny wireless cameras and microphone. The remote control memory tagging and retrieval interface, introduced in chapter 6 should be replaced with a brain to computer interface.

Appendix A Privacy and the machine-based memory manager

Machine-based memory management and privacy are intertwined. When an individual's memories are stored outside his brain it becomes a lot easier for personal memories to be subject to scrutiny by entities other than the individual. Hence once of the first recognizable problems of using a machine-based memory manager is the loss of individual privacy [Schilit *et al*, 2003], [Cheng *et al*, 2004], [Wu, 2006]. However, this is not a good enough reason for not using a machine-based memory manager.

Privacy violations enabled by technology

Even when an individual does not use a machine-based memory manager, current technology has enabled an environment where the individual's privacy is easily invaded. For example, if an individual owns a cellular phone he can easily be tracked through the phone's unique identifier [Eagle, 2005]. Any device which contains a wireless transceiver can be tracked and identified. Small [Small *et al*, 2000] demonstrates location determination from devices connected to a WIFI (802.11) access point. MIT's wireless network [Senseable.MIT, 2005] enables tracking a community of users which use wireless Internet infrastructure in a campus environment. Want [Want *et al*, 2000] points out that object tagging technologies such as Radio Frequency Based Identification (RFID) together with an infrastructure like the Internet make it very easy to identify and track objects and people from anywhere in the world. He urges strongly against the use of RFID to tag and track people as there is a potential for governments and businesses to abuse it.

Communication over the Internet continues to happen in the clear, susceptible to interception and monitoring. With the adoption of devices such as camera-equipped cellular phones, cameras and microphone have entered people's private spaces. Coupled with the Internet, this creates an environment which simplifies the surveillance of communities. Furthermore, installation of surveillance cameras in public areas by

institutions and governments is now ubiquitous. The institute of applied autonomy (IAA) [IAA, 2005] cites that video surveillance overseen by third parties, such as police officers and security guards, tends to target certain groups of individuals such as minorities and women and is often abused. Mann [Mann, 2001] observes that there seem to be orders of acceptability in violation of privacy in society which he lists, ordered from most to least acceptable, as:

- governments spying on people
- establishments spying on people
- establishments spying on establishments
- people spying on establishments
- people spying on the government

The machine-based memory manager as a tool for reverse surveillance

Rather than enabling the violation of individual privacy, the machine-based memory manager can serve as a tool which helps maintain individual rights. Mann [Mann, 2001, 2004] and the IAA [IAA, 2005] argue that individuals who record their external memories can help turn the table against institutionally sponsored surveillance by performing reverse surveillance. The external memories managed by a machine-based memory manager can, for example, be used as evidence in court cases or used to share the knowledge of surveillance activity with other individuals.

Protecting the external memories managed by the machine-based memory manager

Data encryption software can be used to protect personal memories which are managed by a machine-based memory manager from being accessed by entities other than the individual that owns them. However, this type of data protection alone is not enough. Cheng [Cheng *et al*, 2004], Vemuri [Vemuri *et al*, 2004] and Wu [Wu, 2006] argue that even if obscuring protocols and encryption are used for protecting memories, the existence of a record of memories means stored memories are subject to scrutiny from a

third party (the government, private individuals and the law). In such circumstances, individuals may be forced to make available external memories from their machine-based memory manager. To counter this threat to privacy, the individuals that created TrueCrypt [TrueCrypt, 2005] promote the concept of *plausible deniability*. It operates on the premise that even if individuals are suspected of encrypting their data in order to hide it from others, they can deny any knowledge and use of TrueCrypt because the data protected by TrueCrypt cannot be identified as such. It just appears like random data on a storage device.

Appendix B A survey of sensor data classification works

Sensing is ubiquitous amongst biological organisms. Human beings as well as animals rely on their senses for survival in their natural environment. Impairment in any of the vital senses of an animal can result in inconvenience and may even lead to the death of the animal if it is unable to cope with conditions in its natural environment. Sensors play a similar role in machines. Many of the problems which are identified by HCI research are caused by a lack of adequate sensors in machines for sensing user states. For example, on most desktop computers just two sensors (a mouse and keyboard) are available to sense user input, on a mobile device such as a cellular phone, just a keyboard. More recently, microphones and cameras have been added to the list of sensors which machines can utilise, but these sensors are used less frequently than input devices such as keyboards or mice. Machines lack the most basic of sensing abilities: *knowing who their owner is, if and when their owner is about to make use of them or is making use of them,* and doing it *accurately.* Knowledge of this information is most critical in order to offer services to users transparently such as performing power management functions. For example, if machines know that they are not being used by their owner they can turn themselves off at appropriate times to save power.

In context aware computing machines are aware of the states of people and their environment by performing inference on collections of context descriptors. In order for machines to make use of context descriptors, they have to be equipped with suitable sensors which enable them to acquire information from users and their environment.

B.1.1 Categorization of sensors

Currently, sensors are widely used in many fields, for example, in implementations of smart environments, in robotic systems, in consumer devices, in industry as well as academic research. Schmidt [Schmidt, 2002] proposes a categorization of sensors which can be used for determining context into 12 groups as shown in Figure B1. This

categorization includes the human senses (sight, smell, touch, hearing, and sound) as well as physiological sensors but these are not directly specified. The proposed categories are based on sensor characteristics, rather than their usefulness for determining people's and environmental states. In each category a large number of options are available for sensing a particular type of physical phenomenon. Wilson [Wilson, 2005] presents sensors which are used in industry to sense various phenomenon which fall in each of Schmidt's proposed categories. A better categorization of sensors which can be used to determine user context would be a sensor categorization which matches categorizations of context descriptors that have been identified as being useful in determining people's and their environmental states. Bristow's [Bristow *et al,* 2004] context descriptor categorization which includes event, environment, task, person and object context descriptors, is an example.

Sensing Technologies
Light and Vision
Audio
Movement and Acceleration
Location and Position
Magnetic Field and Orientation
Proximity, Touch and User Interaction
Temperature, Humidity and Air Pressure
Weight
Motion Detection
Gas-Sensors and Electronic Noses
Bio-Sensors
Zero-Power Sensors

Figure B1. Schmidt's classification of sensors which can be used to help acquire context descriptors [Schmidt, 2002]

B.1.2 Previous work in sensor data classification

Researchers, in their attempts to investigate context aware applications or new types of user interfaces, develop new techniques by analysing the data which is acquired from sensors. For example, Jonker [Jonker *et al*, 2003] describes sensors which track user location, user motion, visual markers in the environment, user's physical states, speech sensors and visual sensing as being important to sensing context. In order for data from sensors to be used for context determination, it must be processed and features of interest extracted from it through the process of sensor data classification. In this section, a

survey of sensor data classification work which may potentially be useful for determining context is presented.

B.1.2.1 Visual sensor data classification

Visual sensor data classification involves the use of a video camera to capture video frames and analysis of captured frames to identify features of interest. Several types of visual sensor data work have been done in the past including object, location, gesture, eye gaze and activity detection, Figure B2, B3, B4 and B5 show some examples.

Figure B2. Visual sensor data classification: Rekimoto's CyberCode [Rekimoto *et al*, 2000], Nelson's object tracker [Nelson, 2002], Ueoka's object cam [Ueoka, 2004], document tracking [Kim *et al*, 2004], interaction with cellular phone [Rohs *et al*, 2004]

Objects

Kim [Kim *et al*, 2004] illustrates an application of video based document tracking. Kim's system uses a video camera to track document usage on an office desktop and later allow users to browse and search for documents of interest using keywords. Nelson [Nelson, 2002] also tracks objects by analysing frames captured from video. Rekimoto [Rekimoto *et al*, 2000] developed CyberCode a visual tagging system which uses codes printed on paper to identify places and objects using a video camera. SemaCode [SemaCode Corp, 2005] is a similar system which uses the camera in cellular phones to detect visual tags which encode URLs. Rohs [Rohs *et al*, 2004] highlights the usefulness of the camera on a cellular phone as interaction device by showing how it can used to select items from menus printed on paper. Ueoka [Ueoka, 2004] uses a wearable camera which can sense IR in addition to colour to identify nearby target objects. His demonstration shows various objects, which are designated as target objects, being identified from a video frame.

Location

Aoki [Aoki *et al*, 1999] and Clarkson [Clarkson *et al*, 2000] use a wearable video camera to help determine a user's location. They match sequences of live images captured by their wearable camera which describes the trajectory of a user's motion to a database which contains images previously captured during a training phase, to help determine user's location. The resultant location classification algorithms work only for specific cases.

Gestures

Hagan [Hagan *et al*, 2002] uses analysis of video frames to track gestures made by hands. Recognized hand movements are used as a gesture based input interface to a computer. Gestures can be used to perform selection, resizing, zooming and rotation based operations. Hagan identifies gesture-based input as being natural and unobtrusive. Gesture recognition eliminates the need for input devices which require hardware to be worn on the hands.

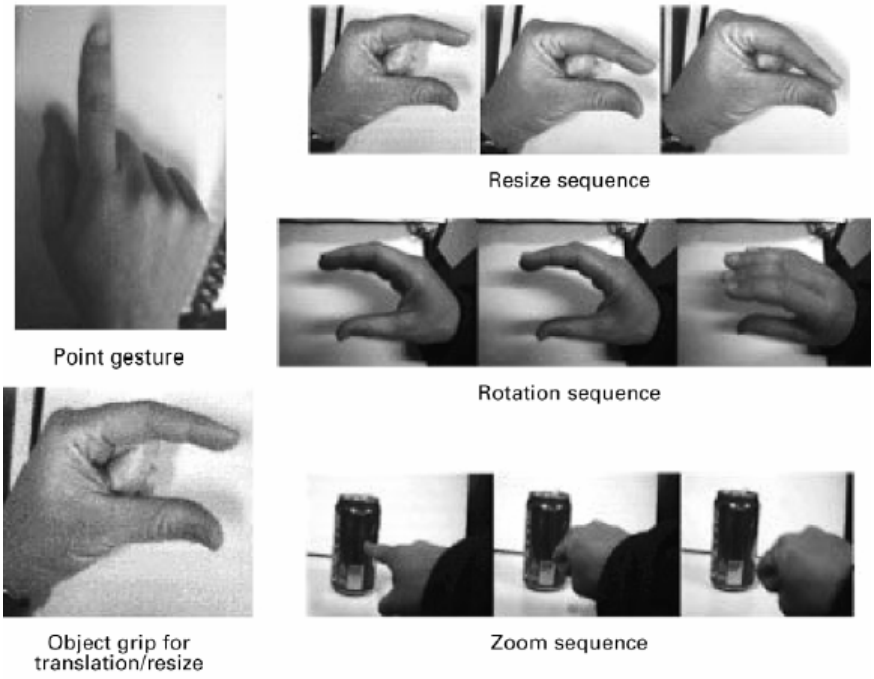


Figure B3. Hagan's gestures [Hagan *et al*, 2002]

Eye gaze detection

Goldberg [Goldberg *et al*, 1999] uses an IR CDD camera-based system to detect eye movements for use as an interface to a computer. Similarly, Betke [Betke *et al*, 2002] presents the video mouse a system designed to assist people with disabilities to interface with computers. Betke [Betke, 2004] also presents a survey of video based HCI interfaces which include face and hand tracking for processing sign language and hand based computer control, communication through blinking of the eye and eyebrows as well as eye gaze detection which is important for detecting user focus. Sumi [Sumi *et al*, 2004] presents a system which comprises of two cameras mounted near the eyes to record what the user sees and one camera which monitors the user's eyes. The system is worn on the head like a helmet and acts as a gaze and direction detector monitoring and recording what the eye is currently focusing on. Sumi has demonstrated its operations by showing how it can be used to create digital forms of real 3D objects.

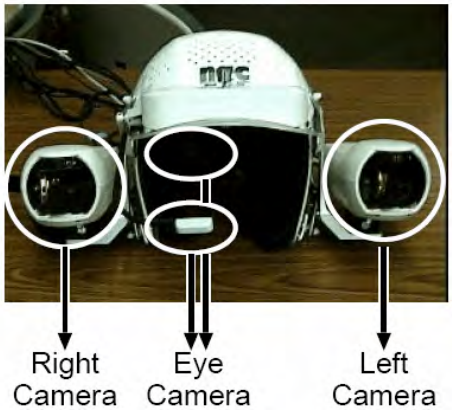


Figure B4. Sumi's wearable camera [Sumi *et al*, 2004]

Activity

Stauffer [Stauffer *et al*, 2000] develop a video based monitoring system for observing moving objects to learn patterns of activity over extended periods of time. Stauffer aimed to identify unusual events and interactions between objects being tracked in a site which is being monitored. He shows his system being used to track cars on a street, and patterns on streets over long hours and claims that the system has been successfully used to track people and objects in indoor environments. Cheng [Cheng *et al*, 2001] uses a camera to detect users' hands and feet. He then overlays graphics on a head mounted display to help guide a person while performing tasks such as playing a piano or dancing. Cheng's main focus is on detection of user context by monitoring a user's hand and feet with the help of a video camera.

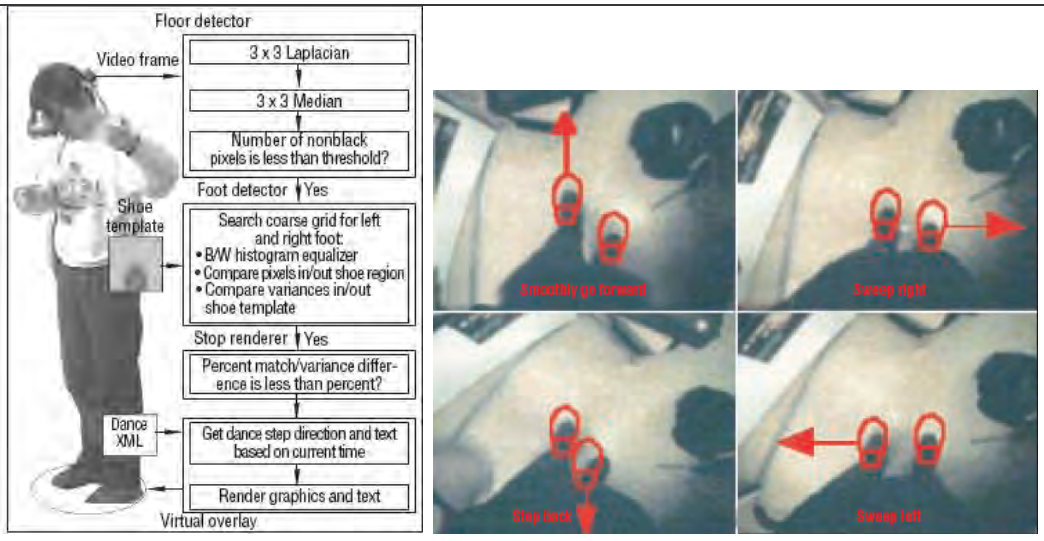


Figure B5. Personal context awareness by analysing video frames [Cheng *et al*, 2001]

B.1.2.2 Physiological sensor data classification

Healey [Healey *et al*, 1998] exploits galvanic skin response (GSR) which involves measuring changes in skin conductivity to monitor users' state. Healey's system captures images by monitoring changes in GSR under the assumption that changes in skin conductivity occur when users are in an excited state. In addition, Healy and Picard [Picard *et al*, 1997] proposed the integration of sensors which measure temperature, blood pressure, heart rate from and muscular electrical activity into a wearable computer to enable the computer to be aware of a user's physiological states. Teller [Teller, 2004] presents a similar system, called SenseWear, for collecting physiological data from users. SenseWear is a commercial product aimed at health care applications. It is equipped with multiple sensors which enable it to take continuous physiological information from individuals.

Figure B6. (1) Wearable ring [Asada *et al*, 2003] (2) SenseWear [Teller, 2004], (3) StartleCam [Healey *et al*, 1998]

SenseWear is mounted on the arm of a person and contains sensors which make physical contact with a person's skin. It can measure user acceleration, heat flux, galvanic skin response, skin temperature as well as time. There are future plans to add a heart rate

monitor. The kinds of information which can be obtained from the collected data include user energy expenditure (calories burned), duration of physical activity, number of steps and sleep/wake states. Asada's [Asada *et al*, 2003] system is also a physiological sensor in a smaller and less intrusive form factor. Figure B6 shows the apparatus used in the above systems.

B.1.2.3 Smell and taste sensor data classification

Sensing of taste and smell is less common than sensing or other features of humans. Toko [Toko, 1998] identifies five qualities of taste (sourness, saltiness, sweetness, bitterness, caffeine) and chemicals which are used to produce them. He suggests taste cannot be determined by just measuring the chemical composition of food stuffs which is done by chemical sensors and asserts that the relationship between taste and chemical composition is not clear. Toko produces a taste sensor which he says is an intelligent sensor which discriminates senses using high level sensing rather than directly measuring the chemical composition of food stuffs. His sensor was used to differentiate food stuffs such as beer, mineral water, coffee and tomatoes. Toko's sensor provides a quantitative, objective measure of taste but not a subjective measure of taste such as *deliciousness*. Wyszynski [Wyszynski *et al*, 2005] notes the lack of investigation of smell sensors. He produce an electronic odor recording and reproduction device and demonstrate their approach by recording and reproducing the smell of citrus fruits.

B.1.2.4 Sound data classification

The affordability and ubiquity of sound sensing devices such as microphones make it easy to use sound sensors. Scott [Scott *et al*, 2005] suggests that sounds made by objects and people can be used to locate people and objects accurately using at least four microphones. Sensing and imaging of objects using sound is exploited by sound sensing systems such as SONAR which is commonly used industrial and military applications. Sensing of more complex sounds such as speech made by humans has been accomplished by speech recognition systems such as CMU Sphinx [CMU Sphinx, 2003], Festival [Festival, 2003] and [Microsoft Speech, 2005]. These systems are almost good enough for everyday use.

B.1.2.5 Environmental sensor data classification

A variety of sensors can be employed for sensing environmental phenomenon such as temperature, humidity, gases, wind speed, altitude etc. Wilson [Wilson, 2005] prevents a survey of such sensors. Schmidt [Schmidt, 2002] highlights how gas sensors such as a carbon monoxide (CO) sensor, can be useful to people.

B.1.2.6 RF based (location, proximity and object) sensor data classification

Location is one of the most used context descriptor. Global positioning systems such as GPS and local positioning systems which use custom RF (Radio Frequency) systems are widely in use. Place Lab [Place Lab2, 2005] is an initiative and a software framework for location based systems. Patel's [Patel *et al*, 2004] ultrasonic line of sight system is an example an indoor location classifier system which is used to locate and identify people and objects. Eagle [Eagle, 2005] uses a cell phone as a device which senses location as well as proximity to collect data from multiple individuals, which he uses to extract what he calls complex social behaviour patterns by observing change of location and proximity of individuals with others over extended periods of time. Small [Small *et al,* 2000] demonstrates the feasibility of user location tracking by analysing wireless LAN (802.11) signal strength. Want [Want *et al*, 2000] and Stanford [Stanford, 2003] highlight the uses of Radio Frequency Based IDs (RFID) based tagging systems whose current industrial use include animal tagging, product tagging as well as access control. Section 3.5.2.1 also considered location data classification systems which use video rather than radio frequency based location identification.

B.1.2.7 User motion sensors

Farringdon [Farringdon *et al*, 1999] proposes a wearable jacket, shown in Figure B7, which is constructed using advanced knitting techniques to form a soft fabric. Stretch sensors contained in the jacket are used to measure upper body and limb movements. Farringdon also proposes a belt worn badge which measures user acceleration which is to be used for recognizing when a user is sitting, standing, lying down and running. Randell *et al* [Randell *et al*, 2000] and Kern [Kern *et al*, 2002, 2003, 2004] also use multiple

accelerometers placed around the body to determine user motion such as walking, running, sitting down and head movement.

Figure B7. Farringdon's wearable jacket containing stretch sensors [Farringdon *et al*, 1999]

B.1.2.8 Document sensors

Document sensors are software based sensors which track and classify various types of electronic documents which are used by people on their electronic devices. On the windows operating system Google Desktop [Google, 2005], Copernic Desktop Search [Copernic, 2005], Blinkx [Blinkx, 2005], [Microsoft, 2005] are examples of desktop electronic document indexers which monitor and index a wide variety of electronic documents. Documents in the real world can be tracked with object sensors described in sections 3.5.2.1 and 3.5.2.6.

B.1.2.9 Multi sensor data classification

Multi sensor data classification is commonly performed by researchers in context aware computing. It involves assembling two or more sensors used to monitor user and environmental activity. Aizawa's [Aizawa, 2001, 2003, 2004] memory capture system consists of a number of sensors (GPS, cell phone, brain wave analyser, gyro and accelerometer for motion estimation) as well as sensor data classifiers for these sensors. His system also senses electronic documents such as email and internet documents. Clarkson [Clarkson, 2002] has demonstrated the extraction of life patterns which are recurrent, recognizable events which are part of people's lives from data recorded by

wearable sensors (audio, 360 degree video and orientation) chosen to match the visual and auditory sensors which people possesses. Clarkson states that he was unable to include smell and taste sensors into his apparatus because the technology was not available at the time he was conducting his experiments. Gemmell [Gemmell *et al*, 2004] has produced SenseCam, a camera which is worn around the neck and contains sensors which record video, light intensity, temperature, motion, orientation and location. His MyLifebits application detects electronic documents. Kwahara [Kwahara *et al*, 2004] uses a combination of sensors to offer situation based services. Her multi sensor system consisted of ultraviolet, brightness, humidity, alcohol, motion and location sensors.

Schmidt [Schmidt *et al*, 1999] presents a design for sensor package called TEA which contains multiple sensors chosen to match the human's senses and environmental sensors consisting of photodiodes for sensing light, accelerometers, passive IR for detecting proximity, temperature and pressure sensors, Carbon Monoxide sensor and microphone. Laerhoven's [Laerhoven, 2002] multi sensor package, shown in Figure B8, has the same goals has the TEA multi sensor package. It is an attempt to create a generic multi sensor board into which different sensors can be plugged. The sensor package is then connected to a computer which analyses raw sensor data produced by analogue sensors such as accelerometers and temperature sensors. It has capacity of reading data from 30 sensors which produce their output in pulse width modulation (PWM) format rather than analogue sensors which produce varying voltages usually in the range of 0-5V.

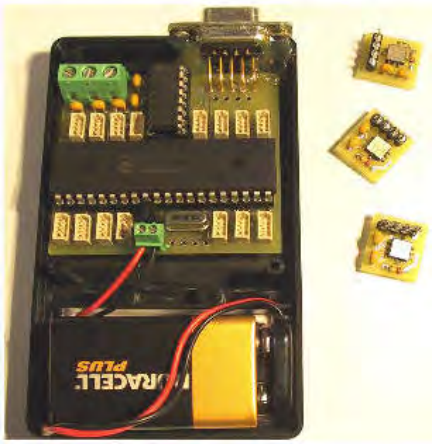


Figure B8. Laerhoven's PWM multiple sensor analogue to digital convertor package [Laerhoven, 2002]

An example of an expensive installation of a smart home environment, costing more than $100,000, is PlaceLab [PlaceLab, 2004],[Intille *et al*, 2005], which is an ongoing smart home experiment containing a large number of sensors which are networked and used to monitor individual behaviour with respect to interaction with ubiquitous digital devices found in people's homes. Sensors distributed around the smart home include pinhole cameras, microphones, touch sensors, biometric and motion sensors. PlaceLab is positioned as a tool which can be used by researchers to study situations in real homes. This is an alternative to laboratory based studies which are more prevalent.

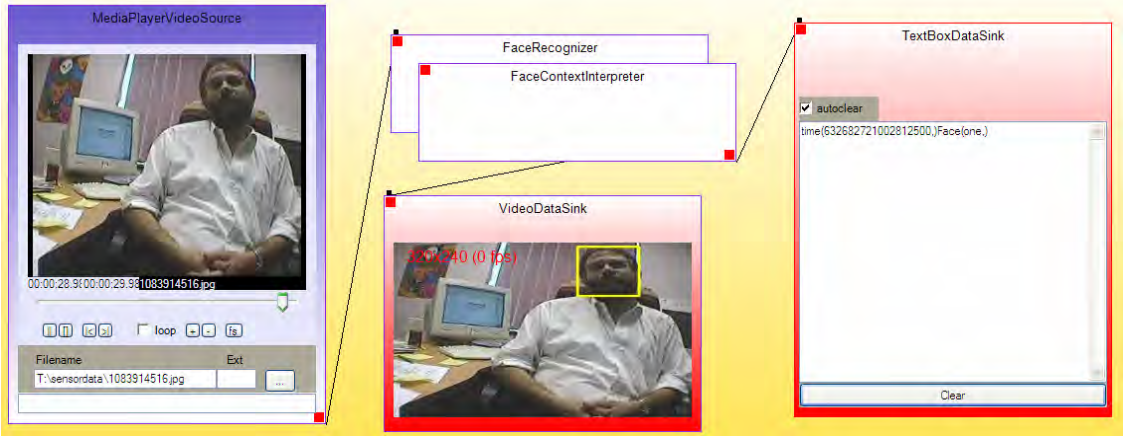
B.1.2.10 Thought sensor data classification

Thought sensor data classification involves creating an interface to the human brain in order to monitor nerve activity with the aim of identifying what users are thinking of. It is hoped that such sensor data classification will eventually enable human to human as well as human-to-machine telepathy. Cybernetic Inc [Cyberkinetics, 2005] as produced a product they call the brain gate which interfaces to the nervous system so that individuals can control devices by just thinking. Gasson [Gasson *et al*, 2004] implants an electrode array into a person's arms in order to establish a bi-directional link between an individual's nervous system and a computer. Using such a setup he is able to control an external prosthetic device using thought. Leuthardt [Leuthardt *et al*, 2004] also implants an electrode on top of an individual's brain in order to allow them to control a computer. Their current implementation enables individuals to control cursor movements on a screen.

Appendix C Testing memory producers individually using the visual programming environment

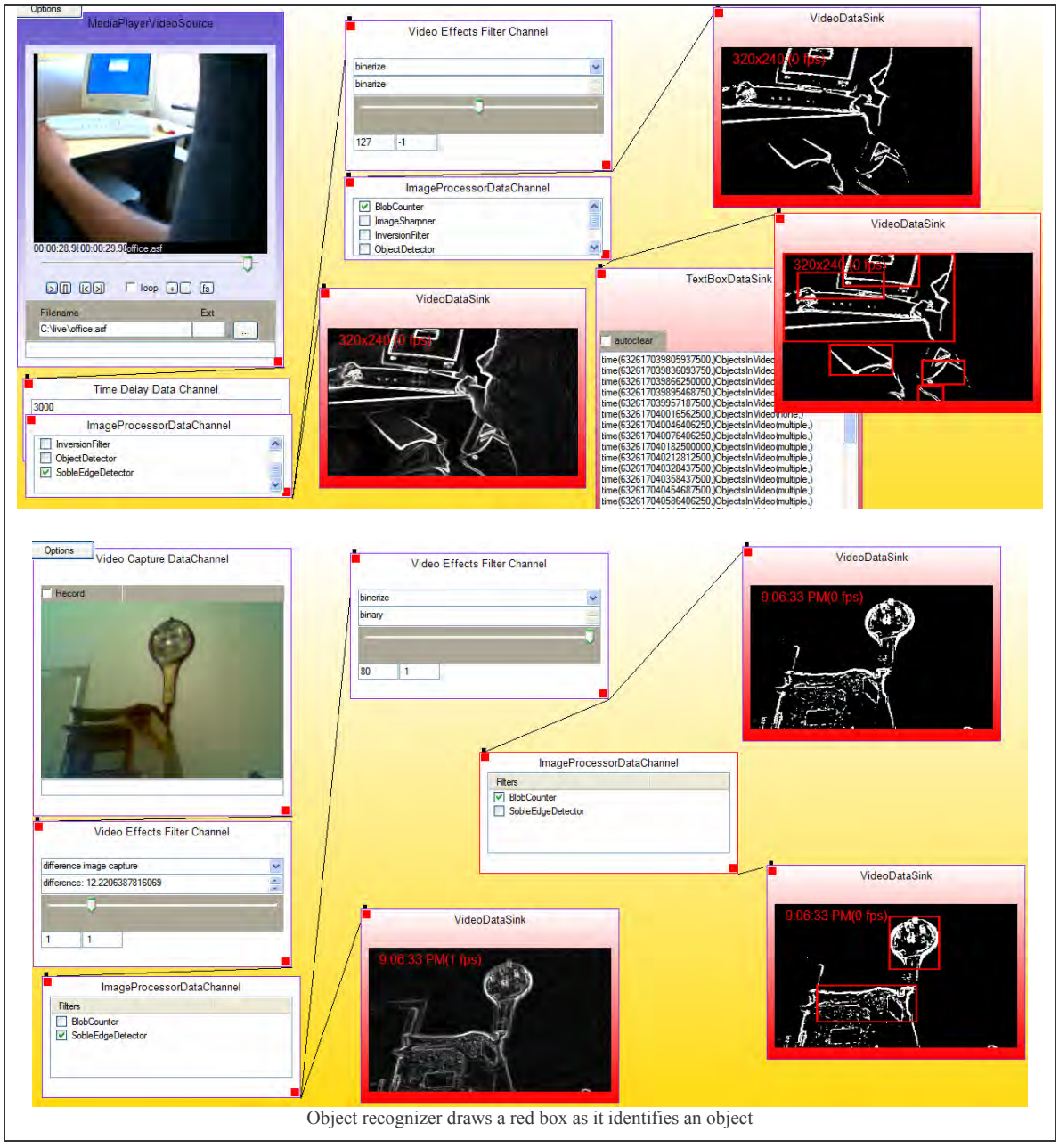
This appendix presents visual examples that show how memory producers are tested individually by attaching one or more data sinks to them. The data sinks show the outputs of each memory producer in real time as they are exposed to external memories. The use of a visual programming environment simplifies the construction of an experimental system and makes it easy to make observations during experiments.

Face recognizer



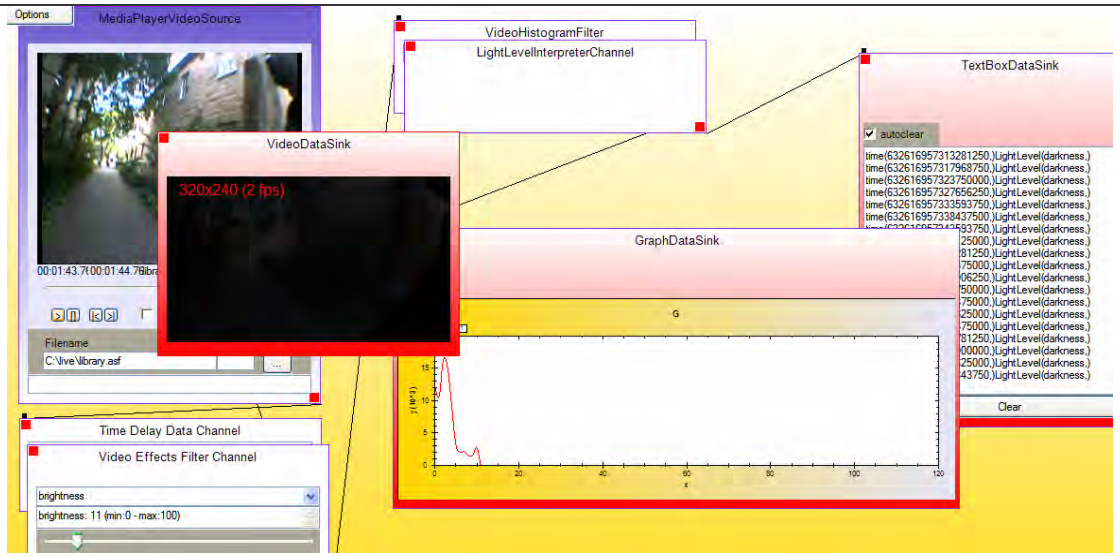
Face recognizer draws a yellow box as it identifies a face

Object recognizer



Object recognizer draws a red box as it identifies an object

Light level interpreter



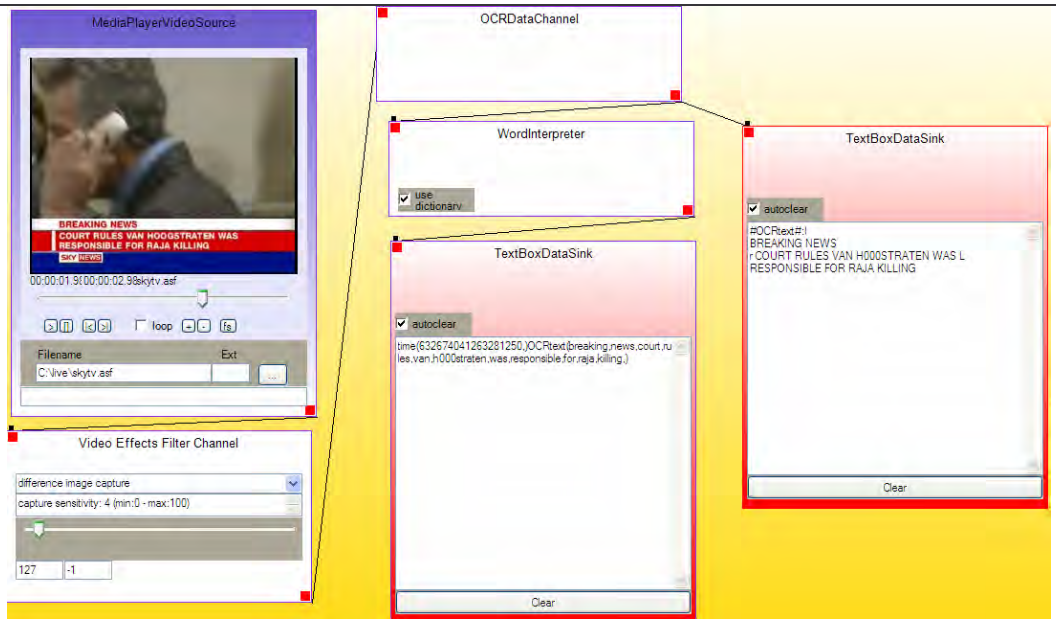
darkness

lowlight

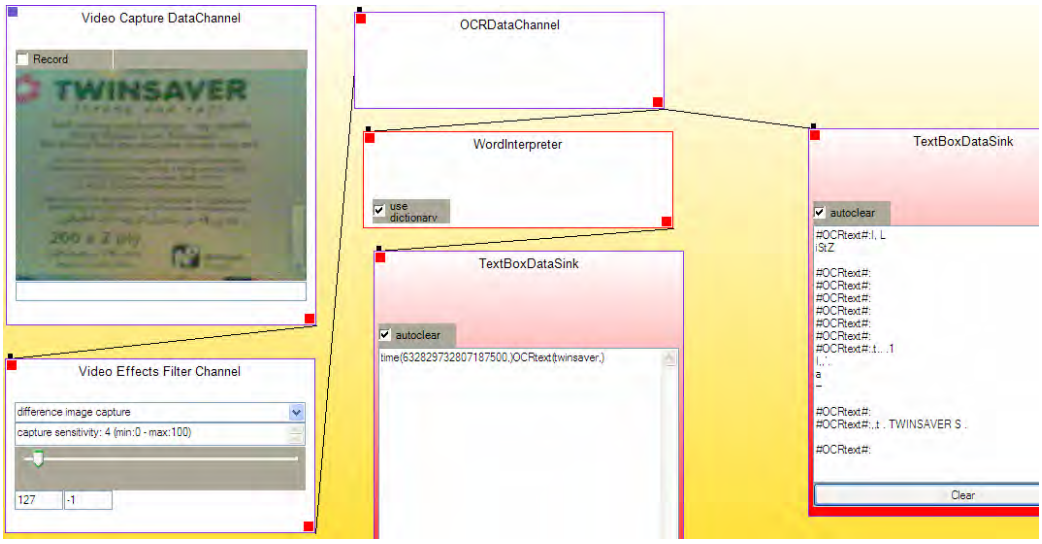
visible

each video source's brightness is altered using the video effects filter channel

OCR based interpreter

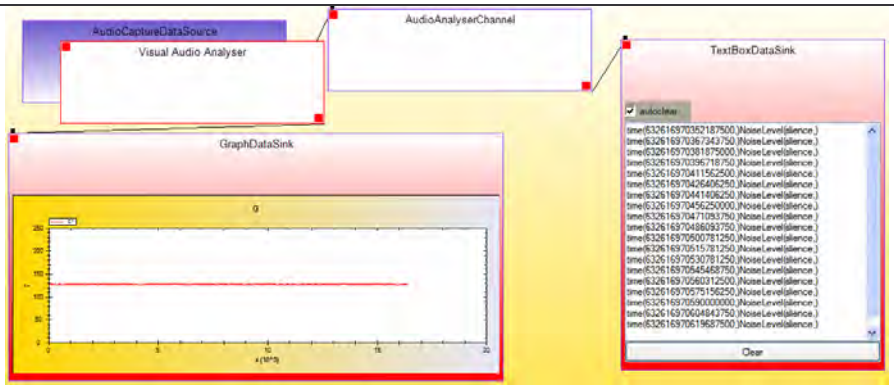


using high quality video (recorded digital video) as input

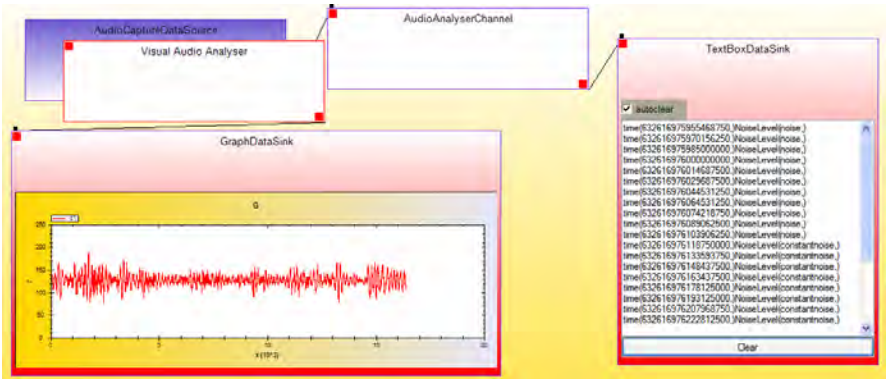


using low quality video (webcam) as input

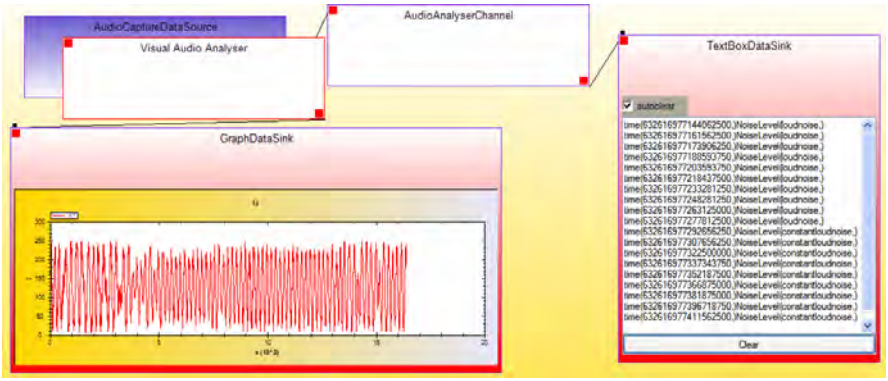
Audio analyser



silence



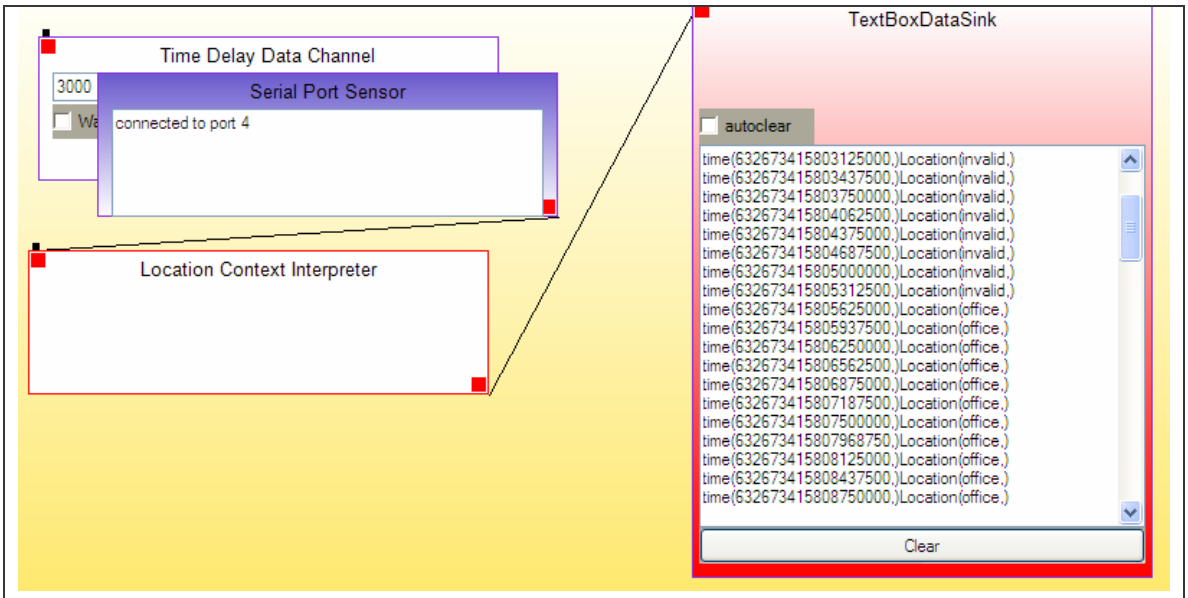
noise



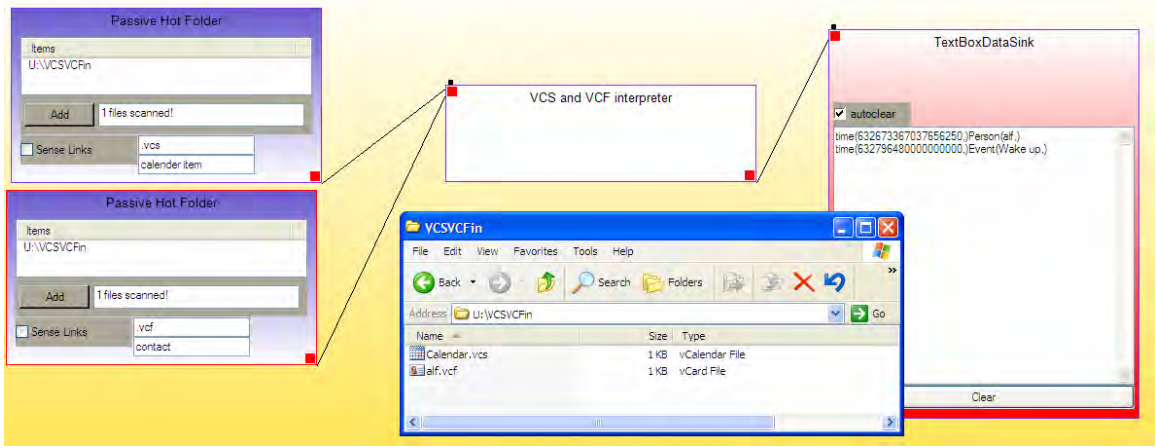
loud noise

The audio analyser being exposed to three types of sounds

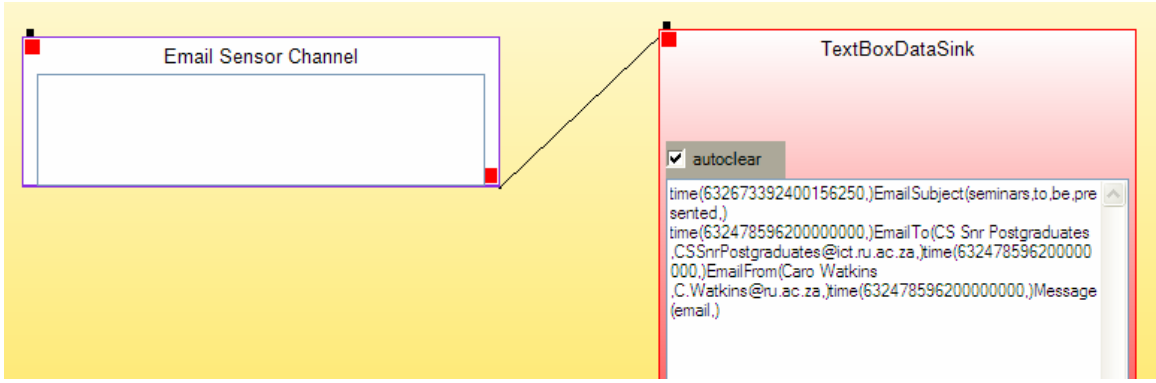
Location, PIM information and email context interpreters



output of location interpreter while moving from indoors (invalid) to outdoors (office), GPS only works outdoors

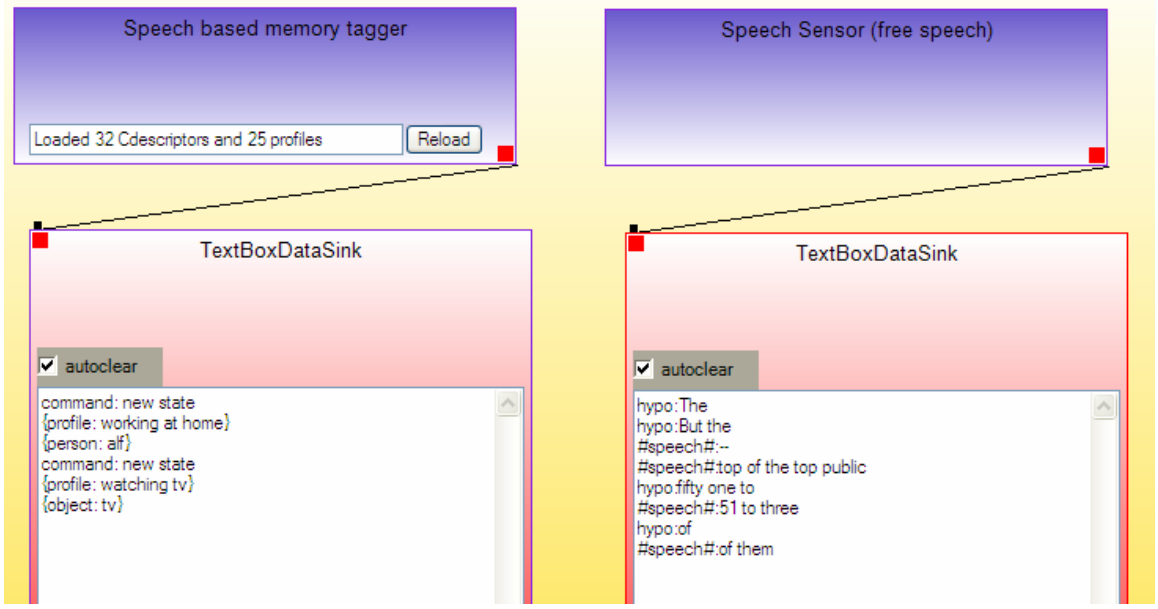


PIM information interpreter identifying context descriptors as items are placed into a hot folder



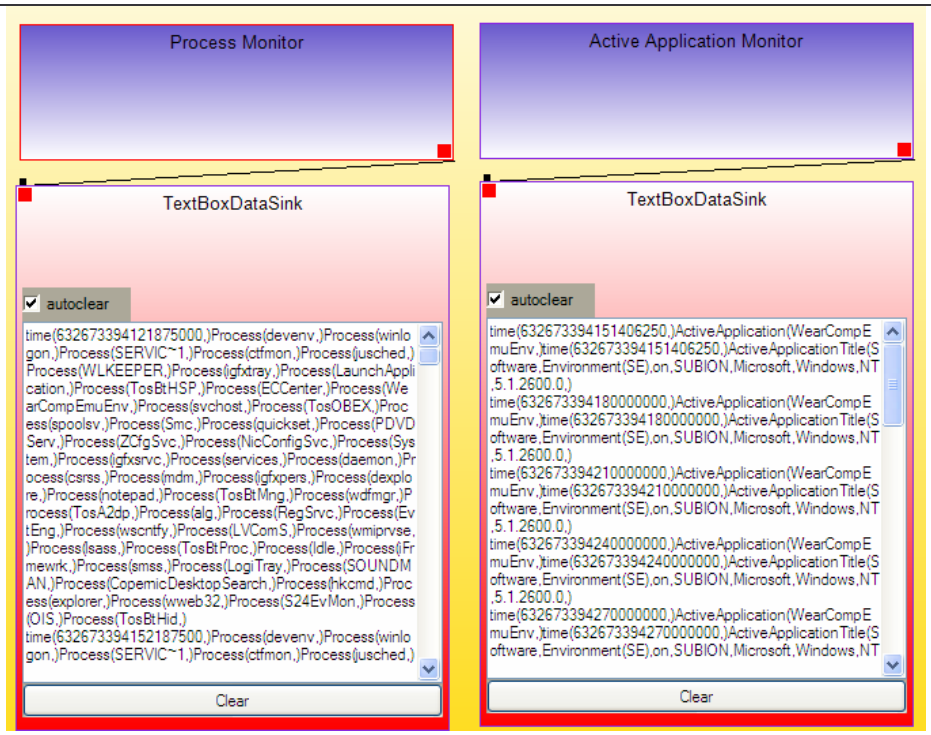
email interpreter extracting context descriptors from an email message as a message is being received

Speech recognizer

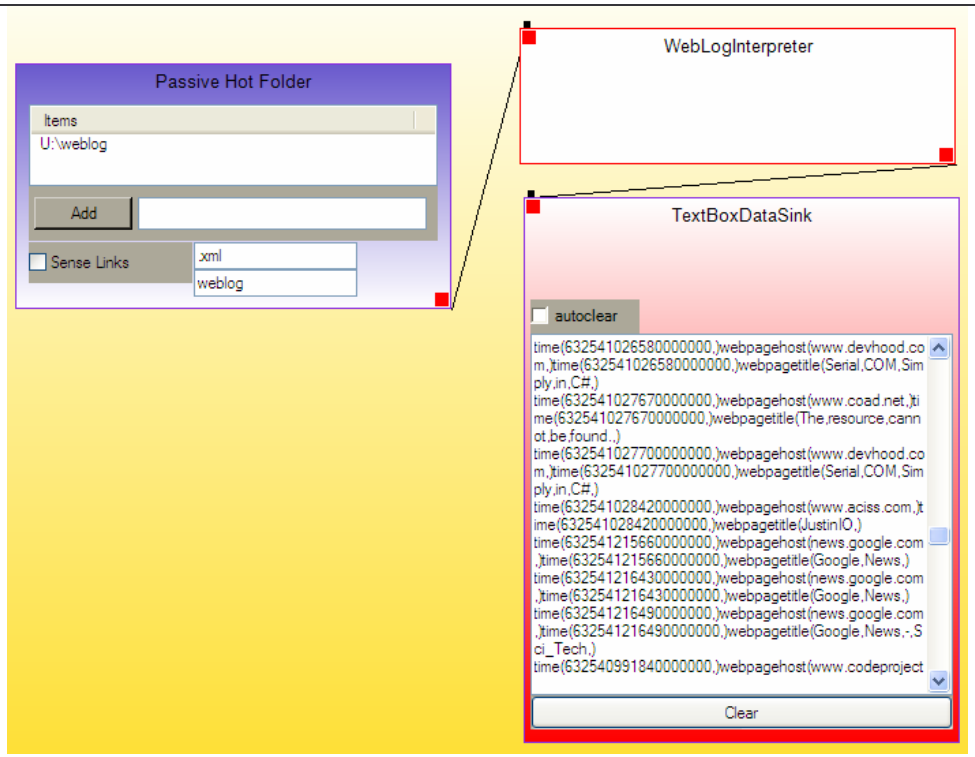


outputs as words are spoken through a microphone.

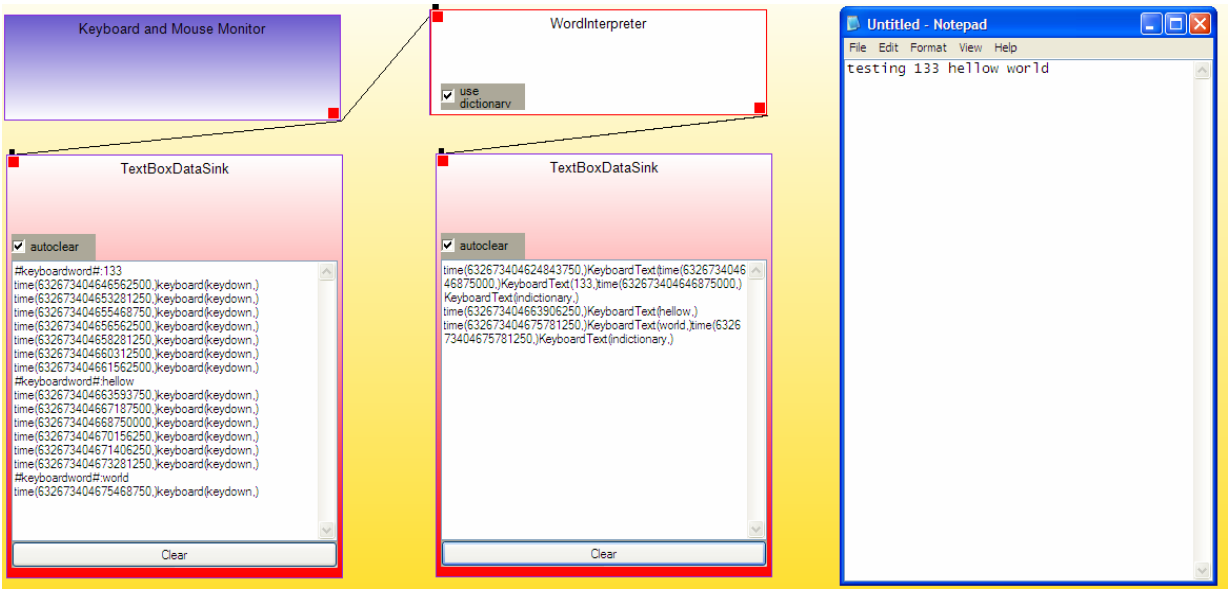
Desktop based context interpreters



Process and active application interpreters identifing currently running processes and applications



Web log analyser interpreting Slogger [Slogger, 2005] log file data from a hot folder



keyboard and mouse interpreters as the application notepad.exe is being used

Appendix D Extending the software environment

In chapter 5 it was described that the software environment can be extended by implementing new black boxes or GUI Views without having to recompile it. The following C# code excerpts illustrate how each of these components can be implemented. Source code listings for black boxes (data channels, data sources and data sinks) and GUI Views is provided.

D.1 Black boxes

A black box, either a data source, data channel or data sink in the software environment is derived from a base class called WComponent which is represented by the following class definition:

```
[Serializable]
public class WComponent
  {
    public string Description;
    [NonSerialized] public Control Parent;
    …
    public virtual void Load()
    {}
    public virtual void UnLoad()
    {}
    public virtual object Clone()
    {
      return null;
    }
    public virtual void ShowPropertyDialog()
    {
      System.Windows.Forms.
      MessageBox.Show("this DataSource has no configurable properties!");
    }
    …
  }
```

Every component has a description that documents what the component does. It is defined at component creation time. A component also contains a reference to the graph node (Control) which data sources, data channels and data sinks can use to render visual representations of themselves when they appear on the graphics canvas provided by the graph manager which is described in chapter 5. All classes are marked as serializable so that the graph manager can serialize all nodes which are part of the graph when it saves

graph models. Non serialized fields are marked as with the keyword *NonSerialized*. The relationship between components and the graph model which is managed by the graph manager is shown in Figure C1.

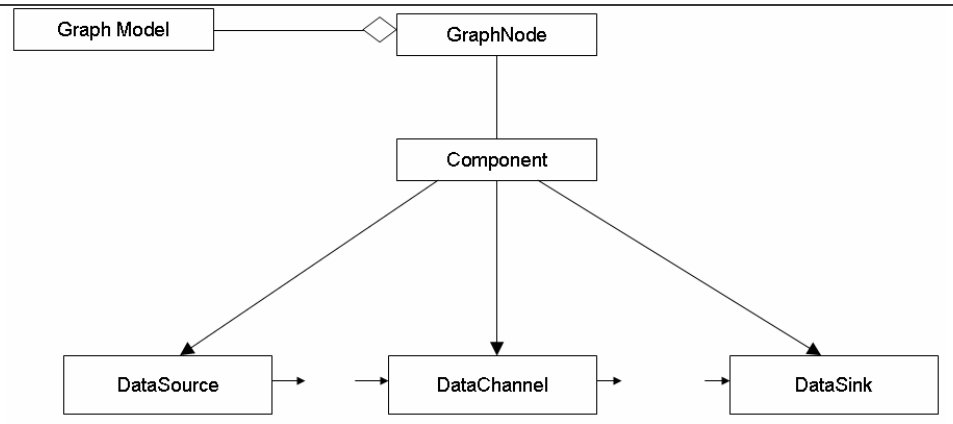


Figure C1. The relation ship between the graph manager, graph nodes data sinks, sources and channels and the graph manager.

When a black box is placed by a user on the graph canvas (as described in chapter 5), the graph manager places a copy of the selected component from its list of components using a component's *Clone* method. The *Load* method is called by the graph manager when a component is placed on the graph canvas and the *Unload* method is called when a component is removed from the graph canvas. A black boxed component can provide a configuration dialog box by overriding the *ShowPropertyDialog* method.

D.1.1 Data sources

All data sources are based on a base class called a DataSource which is represented by the following class definition:

```
[Serializable]
public class DataSource:WComponent
    {
      [NonSerialized] ArrayList dataSinks;
      …
      public virtual void PutToSinks(object data)
      {
      }
      public virtual object GetSample()
```

```
      {
        return null;
      }
      …
    }
```

Data Sinks need to override the *GetSample* method which needs to return the kind of data the data sample produces. This data is passed to data sinks to check if they can process it when a connection is created between a data source and data channels/data sinks. Data sources/data channels needs to call the *PutToSinks* method when they want to push data to data sinks which are attached them.

D.1.1.1 A Data Source Example

The following code example shows an implementation of a sensor data source which produces a clock data stream:

```
[Serializable]
public class ClockDataSource:DataSource
    {
      [NonSerialized] PeriodicThread thread; //runs a piece of code periodically

      public override void Load()
      {
         thread = new PerioicThread(new MThread.MThreadStart(ThreadFunc));
         thread.Interval = 1000; //every second
         thread.Start("ClockDataSource"); //named thread
      }
      Public override void UnLoad()
      {
         thread.End(); //cleanup
      }
      public override object GetSample()
      {
        return DateTime.Now.ToString();
      }
      public override object Clone()
      {
        return new ClockDataSource();
      }
      void ThreadFunc()
      {
          DateTime n = DateTime.Now;
          this.PutToSinks(n.ToString());
      }
    }
```

D.1.2 Data Channels

All data channels are based on a base class called a DataChannel which is represented by the following class definition:

```
[Serializable]
public class DataChannel:WComponent
    {
      [NonSerialized] ArrayList dataSinks;
        …
      public virtual void PutToSinks(object data)
      {…}
      public virtual object Transform(object data)
      {}
      public virtual bool CanConsume(object data)
      {
        return false;
      }
      public virtual object GetSample()
      {
        return null;
      }
      …
    }
```

Data channel implementers need to override the *CanConsume* method to indicate whether they are able to process a specific type of data. This method is called by the graph model manager when a connection between a data source and data channel is made. The *Transform* method performs the data transformation function of a data channel. Inside it data channel implementers need to call the *PutToSinks* method when they want to push multiple data items to data sinks which are attached to them asynchronously. A data channel can return modified data if it only performs a single transformation function.

D.1.2.1 A Data Channel Example

The following code example shows an implementation of a data channel which modifies a clock data stream:

```
[Serializable]
public class ClockDataChannel:DataChannel
    {
      public override object GetSample()
      {
        return DateTime.Now.ToString();
```

```
    }
    public override object Clone()
    {
      return new ClockDataChannel();
    }
    public override bool CanConsume(object data)
    {
         try{
             DateTime val = DateTime.Parse(data as string);
          return true;
         }
         catch {}
         return false;
    }
    public override void Transform(object data)
    {
         try{
            DateTime val = DateTime.Parse(data as string);
            Val+=TimeSpan.FromDays(1); //transform the data
            this.PutToSinks(val.ToString()); //asynchronous message passing
            //return val.ToString(); //synchronous
         }
         catch {}
         return null;
    }
  }
```

D.1.3 Data Sinks

All data sinks are based on a class called a DataSink which is represented by the following class definition:

```
[Serializable]
public class DataSink:WComponent
   {
     …
     public virtual void PutData(object data)
     {}
     public virtual bool CanConsume(object data)
     {
       return false;
     }
     …
   }
```

The *PutData* method is called for every data sample which a DataSink needs to process. The CanConsume Method serves the same function as in DataChannels.

D.1.3.1 A Data Sink Example

The following code example shows an implementation of a data sink which saves a clock data stream:

```
[Serializable]
public class LogFileDataSink:DataSink
    {
      [NonSerialized] StreamLogFile logFile;
      …
      public override object Clone()
      {
        return new LogFileDataSink();
      }
      public override bool CanConsume(object data)
      {   // data sink accepts only clock data samples
          try {
              DateTime.Parse(data as string);
          }
          catch {
            return false;
          }
          return true;
      }
      public override void PutData(object data)
      {
          try{
               DateTime val = DateTime.Parse(data as string);
               val+=TimeSpan.FromDays(1); //transform the data
               logFile.Write(data.ToString();
            }
           catch {}

       }
    }
```

D.2 GUI views

GUI Views are derived from a single class called a BaseComponentManagementView which is represented by the following class definition:

```
public class BaseComponentManagementView
  {
    public string Description=" ";
    ArrayList componentViews = new ArrayList();
    public BaseComponentManagementView()
    {
    }
```

```
    public virtual void ShowSubView(int id)
    {
      ….
    }

  }
```

A GUI View can contain one or more Forms which represents GUIs to the facilities which a GUI View offers.

D.2.1 An Example of a GUI View

The following code example shows an implementation of a GUI View which provides GUI to two facilities:

```
  public class DefaultSensorManagementView: BaseComponentManagementView
  {
    public DefaultSensorManagementView()
    {
      this.Description = "MyGUIView";

      MyForm f = new MyForm();
      f.Text ="This is a GUI for facility1";
      this.componentViews.Add(f);

      MyForm2 f2 = new MyForm2();
      F2.Text ="This is a GUI for facility2";
      this.componentViews.Add(f2);

    }

    override public void ShowSubView(int id)
    {
      Form f = this.componentViews[id] as Form;
      f.MdiParent = ResourceManager.SensorsMdiParent;
      f.Visible = true;
      f.BringToFront();
    }
}
```

When compiled and added to the software environments GUI Views it will appear as one of the facilities in the software environment's GUI View browser.

Appendix E Design considerations prior to the implementation of the machine based personal memory manager

This appendix provides a discussion of the design considerations which were made prior to the implementation of the machine based personal memory manager (described in chapter 6) with a focus on the hardware platform, the software platform and the context aware file system (CAFS). The general design which was made is that, because this is the first instance of an implementation of the machine based personal memory manager, a prototype which operates according to the model (outline in chapter 3) but which does not necessarily make use of the best available components is produced. The goal was to create a prototype which simple to develop, effective and can serve as an experimental platform.

E.1 Design considerations in the implementation of the hardware platform

The machine based personal memory manager makes use of commonly available consumer electronic components which make the experimental platform straight forward to reproduce for further experimentation. Components can be easily upgraded when more advanced versions become available and can also be easily replaced in case of failure. The components are more easily understood than specialized hardware as they are based on industry standards. Most of the device drivers and software libraries which are needed to control the hardware components already exist. A weakness of the hardware platform is that it is very visible (large) and requires a lot of user attention to operate. It does not make use of cutting edge, miniaturized components (such as high resolution wearable wireless sensors) which would have made the device easier for an individual to manage. It does not make use of hardware which is specifically manufactured for life long use. It for example uses a magnetic storage device (hard disk) with a life rating of

decades, for storage, rather than an optical storage medium which has a life span of 100s of years. Consumer level optical storage devices are currently more cumbersome to make use (are slow, limited in storage capacity) than magnetic storage devices. The platform does not specifically address the issue of efficient and long term power use. In the future, these and other shortcomings are expected to be addressed as the technology advances.

E.2 Design considerations in the implementation of the software platform

The chosen approach to development of the software platform of the machine based personal memory manager has similar benefits as the hardware platform in terms of its use of a general purpose computing architecture which is complimented by various industry standard operating systems and software development tools. This makes the acceptance of the experimental platform by third parties easier. Researchers which do not normally have access to specialised hardware, software and knowledge on how to create such a platform will be able to conduct practical research related to personal memory support and augmentation. The use of a visual programming paradigm further reinforces the latter view. An alternative software development approach (which would have been dictated by the choice of hardware such as specialised embedded systems) would have been to develop a highly efficient machine based personal memory manager (in terms of its use of processor, memory and power resources) using embedded software development tools.

In respect to the architecture of the software platform, a data flow based system made up of black boxes, several advantages are gained over traditional monolithic architectures. Sub systems of the machine based personal memory manager can easily be developed and maintained. They can also be individually replaced/maintained without interfering with the operation of the system as a whole, a feature that is important to a lifelong personal memory manager which should run continuously without interruption. Another advantage of the chosen architecture, which needs to be explored in the future is, the

support for parallelization of data processing. Because components of the system communicate by exchanging data, they can be made to execute on systems which contain several processors.

The software platform is designed to be run on a general purpose computer which accompanies a person everywhere. This approach to the implementation has the benefit of storing the personal memories of a person physically close to the individual. In this configuration, the individual is responsible for the physical safety of their personal memories. An alternative implementation strategy which could have been adopted is a networked architecture for the core of system, by moving the computing platform of the machine based personal memory manager to several remote locations. In this configuration, a person would only have to carry sensors and memory consumption interfaces. The processing requirement of the system would be offloaded to the network. The physical security of the personal memories would also be offloaded to a third party. The several instances of a personal memory manager would serve as backups should a single instance fail. The disadvantage of such an architecture is the chance of loss of control of personal memories and the system from the individual. This can happen if the network is taken over by an untrusted third party or in the event of a cataclysmic event which renders the network useless (such as an asteroid striking the Earth, an autonomous system would be useful to survivors of such an event).

E.3 Design considerations in the implementation of the context aware file system (CAFS)

Chapter 6 (section 6.3.1.1) described the implementation of the context aware file system (CAFS) on top of a hierarchical file system. This section relates the requirements of the CAFS to its implementation, provides motivations for choosing the current implementation medium for the CAFS and also identifies possible alternatives, their advantages and disadvantages.

The requirements of the CAFS are specified in chapter 4 (section 4.4.1) as:

1. enable addressing of memory fragments using context descriptor states
2. maintain relationships between context descriptor states and memory fragments with respect to their time of formation
3. provide life long storage, do not discard memory fragments

Further, as part of a useful implementation, this implementation aims to meet the following requirements:

A. Enable fast context descriptor state to memory fragment mappings (less than 1 ms desirable, which would mean hundreds of mappings can be performed in less than a second as would happen if hundreds of context descriptor states are used for retrieval. Normally, a user makes use of a few context descriptor states (<10) for retrieval, but this could increase if more complex means of context based memory fragment retrieval that make use of hundreds of context descriptor states are developed in the future.
B. as this is a life long storage system, reduce disk space use where possible

E.3.1 Relating the requirements of the CAFS to the implementation

Because this is a first instance of an implementation of the CAFS, a prototype that meets the requirements described in section E.1 but which does not necessarily use the best available solution to address the requirements is produced.

The current implementation meets the requirements outlined in E.3 as follows:

Requirement 1: enable addressing of memory fragments using context descriptor states (provide a mapping from multiple context descriptor states to memory fragments)

provided solution: 6.3.1.1 (subsection: context descriptor layer) and section 6.3.1.3 (subsection: memory fragment retrieval stage 1)

motivation for meeting requirement 1: Sets which are represented as bit arrays and set operations which are represented as logical operations on bit arrays are used to perform mapping from context descriptor states to quanta (which contain memory fragments). Bit arrays are a common implementation medium for sets.

motivation for meeting implementation requirement A (fast access): logical operations with bit arrays can be performed fast. Should be performed in hardware where the implementation supports such operations. Current implementation is dependent on the efficiency and integrity of the C# BitArray object. Prior to implementation, test operations (bit modification, logical operations) on 65KB C# BitArrays were observed to meet implementation requirement A.

motivation for meeting implementation requirement B (space use): A single bit is used to represent a quantum pointer for entries in the context descriptor layer and the time of occurrence of the state within a quantum, for context descriptor states stored in a quantum. Prior to implementation, storage space use of a bit array which can hold quanta pointers (for a single context descriptor state) spanning a 10 year period was observed to be 65KB when the temporal size of a quantum is set to 10 minutes. Thus, using collections of bits to represent quanta pointers makes efficient use of storage space. This size is small, considering the time period, the bit array is used to represent. Disk space use could be improved by enabling the compression of bit arrays at the expense of processor cycles (Tests prior to implementation show storage space to reduce to less than

1KB when gzip compression is used). However, a decision was made not to use compression at the expense of processor cycles.

Requirement 2: maintain relationships between context descriptor states and memory fragments with respect to their time of formation

provided solution: 6.3.1.1 (subsection quanta file system layer) and section 6.3.1.3 (subsection memory fragment retrieval stage 2)

motivation for meeting requirement 2: A directory on a hierarchical file system is used to represent a quantum. This implementation medium provides a physical set-like container which contains memory fragments and context descriptor states. It represents a simple mapping from the abstract concept of a quantum to a physical representation.

motivation for meeting implementation requirement A (fast access): The chosen hierarchical file system provides fast access to the contents of a directory.

motivation for meeting implementation requirement B (space use): Context descriptors in a quantum are stored as directories and log files within this directory contain context descriptor states for this quantum. The reason for the later representation is to allow for context descriptor state to memory fragment mapping with a precision of 1 second. This representation would not be necessary if a global quantum size of 1 second is chosen. However, use of a global quantum size of 1 second rather than a 10 minute quantum size would considerably increase the size of the context descriptor layer (the bit array used to represent a context descriptor state for a 10 year period would increase from 65 KB to 3900KB). For a life long storage system, increasing disk space use by a factor of 60 is significant.

Requirement 3: provide life long storage, do not discard memory fragments

provided solution: CAFS (section 6.3.1)

motivation for meeting requirement 3: A component of the model presented in chapter 4

motivation for meeting implementation requirement A (fast access): The chosen hierarchical file system provides fast access to the contents of items in a directory.

motivation for meeting implementation requirement B (space use): The implementation does not currently address the issue of reduced space use with regards to memory fragments in the quanta file system layer. Use of current standard digital data encoding schemes for encoding memory fragments (which may address the issues of reduced space use) is assumed. The chosen file system can accommodate new directories and contents of items in directories spanning over a long periods of time as long has the hardware on which the file system resides contains required space for storing new memory fragments and context descriptors.

E.3.2 Alternative implementation strategies for the CAFS

The following are some alternative implementation strategies for the CAFS.

A flat file database

A flat file database is a primitive implementation medium for the CAFS. This approach would require unnecessary re-implementation of mechanisms for adding, removing and insuring the integrity of memory fragments and context descriptors in the file system. The CAFS network architecture would have to be mapped into a linear storage representation.

A relational database

A relational database would be a useful medium for implementing the CAFS as it provides a standard query language (SQL) for managing items stored in a database. It also provides standardized data integrity and security features but needs optimisation for speed and storage use as part of a life long storage system. The CAFS which has an inherently networked architecture would have to be mapped into a relational model.

An object oriented database

An object oriented database would provide the benefits of representing memory fragments and context descriptors as objects which are related to each other through complex data rather than just text based context descriptors but is difficult to optimise for speed and reduced storage space use. An object oriented database would introduce a layer of complexity into the CAFS which may not be desirable.

Artificial neural networks (ANNs)

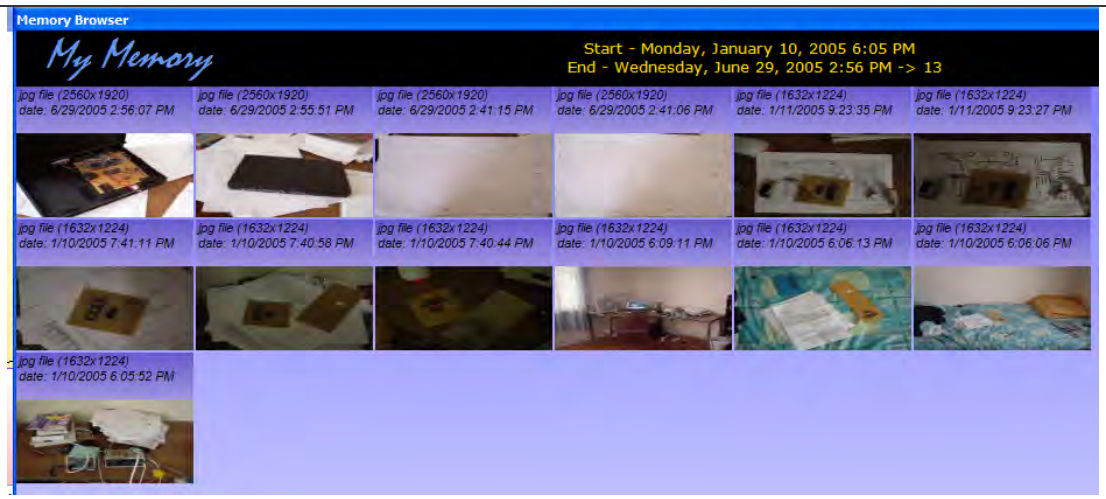
Two ANN based CAFS implementation strategies need further investigation as alternatives to all of the previously considered database implementation mediums.

1. Use of an artificial neural network to represent the context descriptor layer only
2. Implementing the entire CAFS as an artificial neural network but addressing the limitations of biological systems (forgetting - memory loss).

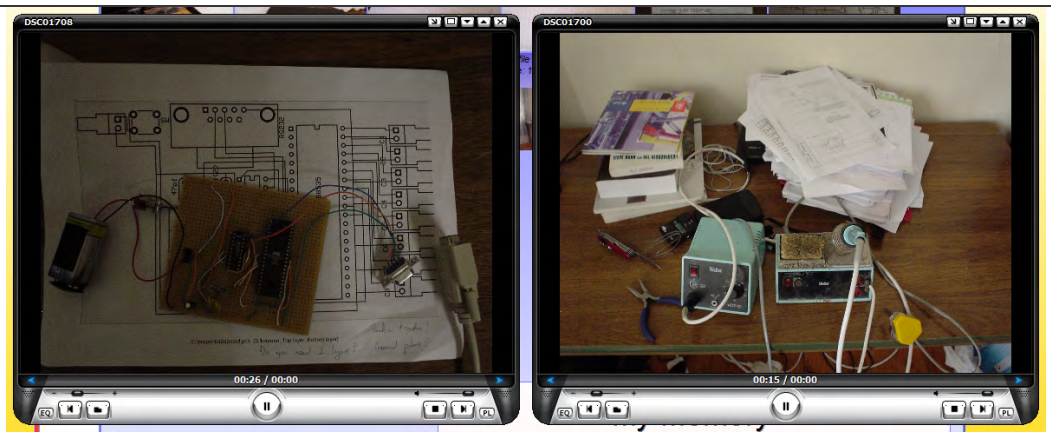
Artificial neural network based implementations use a fixed size memory (can be calculated prior to implementation). They are easier to implement in hardware but can be more complex develop. This approach requires further research: network identification, training strategies, learning functions. In terms of developing a working a prototype in the short term, alternative 1, seems plausible.

Appendix F Eight channel analogue to digital converter

In this work an eight channel analogue to digital (a2d) converter was constructed from scratch. The motivation for creating the a2d converter was to attach sensors that sample data from the real world to it. The constructed a2d board is shown in Figure F1. Unfortunately, the board could not be used in experiments because it is too large and fragile. A description of the process of constructing the board is presented here so that it can be improved in future. A similar a2d converter with more data channels is provided by [Laerhoven, 2002]. Laerhoven's board uses pulse width modulation (digital pulses) to communicate with sensor devices attached to it whereas the a2d board constructed in this work uses voltage changes to communicate with sensors that are attached to it, i.e. it is basically an eight channel digital voltmeter.



Visual memories captured while working with the a2d board



The a2d board, the soldering iron and equipment used to construct the board

The final a2d board enclosed in a VHS cassette holder

Figure F1. Visual Memories of the a2d board

F.1 Board construction

The construction of the a2d board took a long time complete since the author is not an electronics expert. The board was created though a series of steps:

1. Components for the board had to be identified, these include a micro processor chip. Atmel's AT90S8535 8 bit chip [Atmel, 2003] was used. It has a 10 bit a2d conversion facility which means it can express voltage changes as numbers in the range(0-1023). All of the components used for constructing the a2d board are shown in Figure F2. The consist of the AT90S8536 microcontroller, an RS232 connector, 8 connector with +,- and GND, a MAX233 chip for serial com, two 47pf capacitors, one 1mf capacitor, an 8mhz crystal to provide a clock source for

the microcontroller, an LM7895 for reducing voltage from a 9V battery to 5 volts which the board uses and a switch for turning the board on/off.

2. The micro code for the Atmel chip had to be written compiled and uploaded to the chip. Stang's AVR code [Stang, 2002] was used as base to write an interrupt driven a2d conversion microcode. The code was then compiled using WinAVR [WinAVR, 2004] and uploaded to the microchip using PonyProg [Lancos, 2003] and Atmel's stk200 development board.
3. A schematic of the circuit board had to be designed. TinyCAD [TinyCAD, 2004], which is a freely available schematic design tool was used.
4. A PCB layout of the board had to be created using a software tool. ExpressPCB [ExpressPCB, 2004] was used to create a PCB layout.
5. Once a schematic and PCB layouts have been designed, these along with a list of components can be sent to professional board manufacturers. The schematic and PCB layouts are shown in Figure F2, F3 and F4. In this work the components where placed on bearer board and manually soldered using soldering lead and iron which is a tedious process.
6. The board is then connected to a PC though the PC's serial port connecter. Digital data from each of the data channels is streamed over a serial line to the PC. The data is encapsulated in packets each containing the values of the converted analogue signal and the channel the value came from.
7. Eight mini jack connectors are introduced in the a2d converter board as connectors for sensors to make it simple to plug sensors in and out of the board.

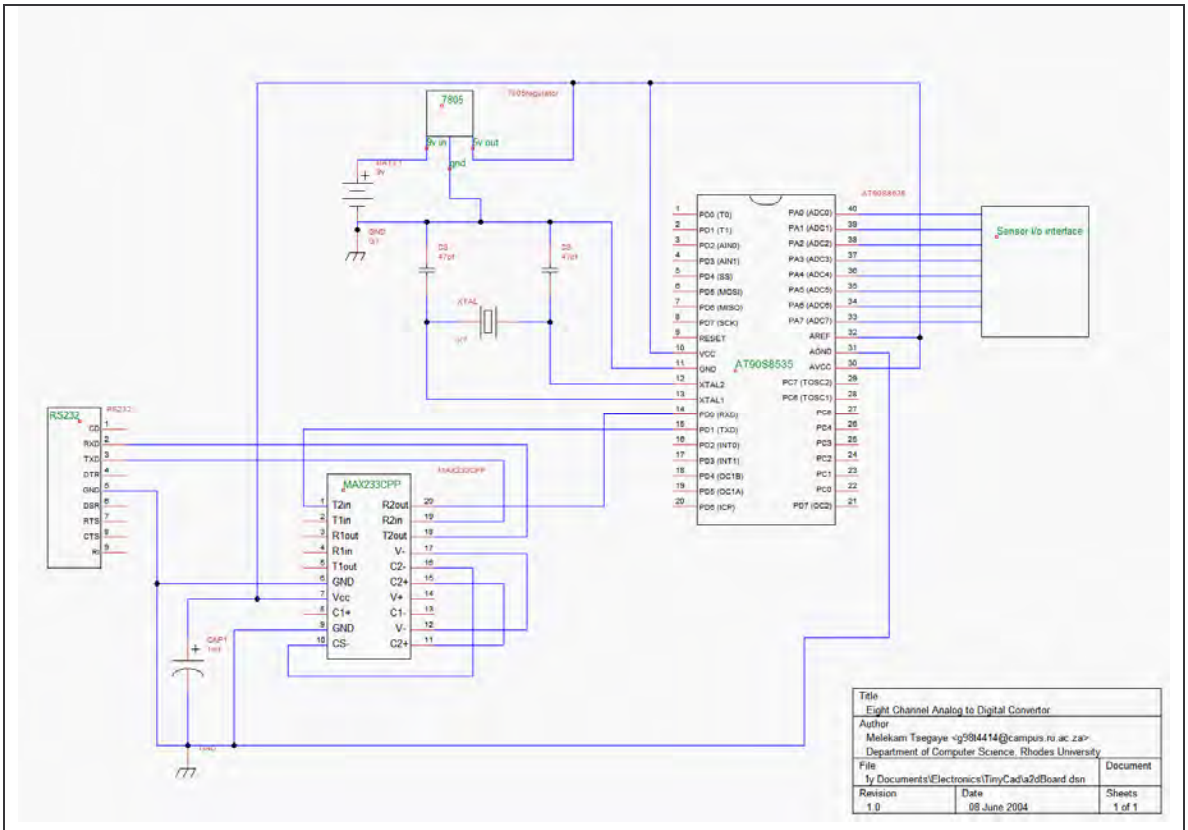


Figure F2. Schematic diagram for the a2d board

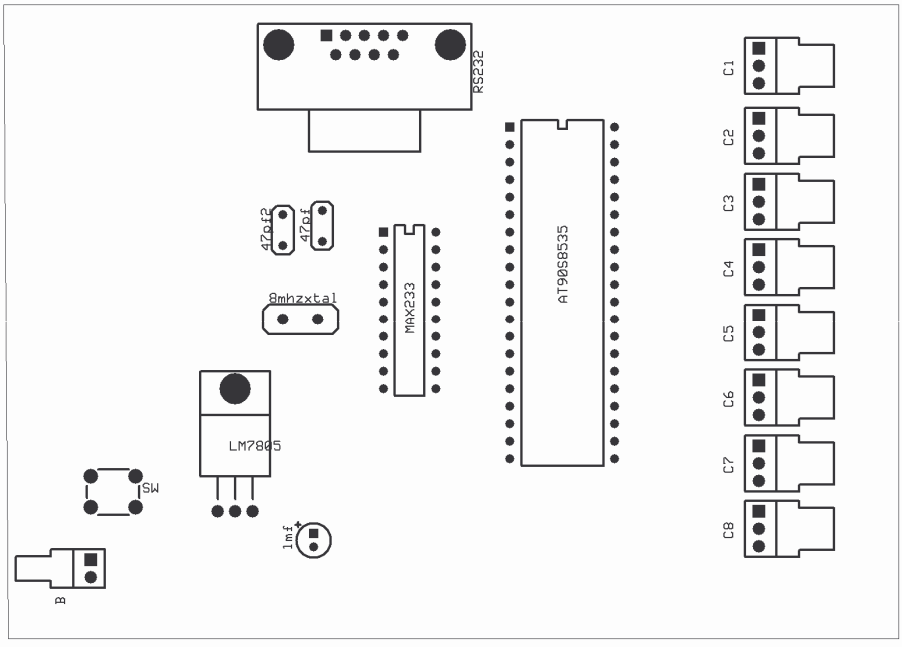


Figure F3. PCB layout of the a2d board showing all the components that were used

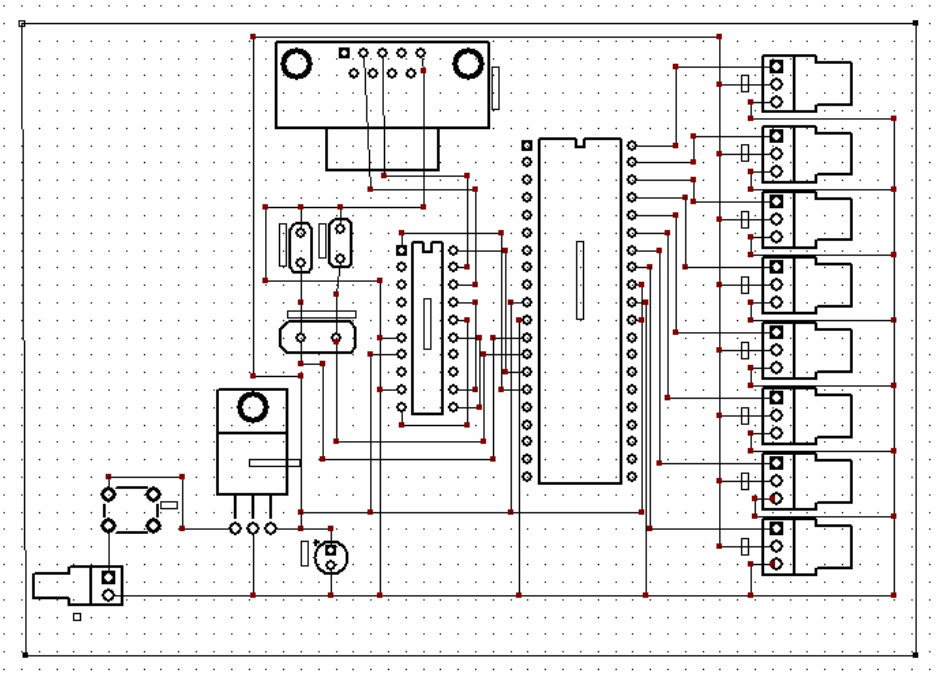


Figure F4. PCB layout showing the connections between the components

Glossary of key terms

BCI (Brain to Computer Interface)

Artificial device which can read a person's thoughts. Can be invasive (directly implanted into the nervous system) or non invasive (monitor electro-magnetic waves emitted by the brain)

Black box

Component with a 2D visual representation which represents the basic building block of applications.

Central executive

A subsystem of memory which controls and coordinates the human memory system.

Cognition

The psychological result of perception, learning and reasoning.

Context aware file system (CAFS)

A file system that stores memory fragments and context descriptors while maintaining a mapping from context descriptors to memory fragments and context descriptors to context descriptors.

Context awareness

An ability which applications have for determining context by considering a number of context descriptors.

Context descriptor layer

A storage layer which contains a collection of context descriptor states that point to collections of memory fragments thereby playing the role of tagging memory fragments. (enables the mapping from context descriptor states to quanta).

Context descriptor state

The members of the set of states of a person and their environment (i.e. context).

Context descriptor

A class of context descriptor states.

Context interpreters

Entities that take as input raw sensor data and/or collections of context descriptors and produce as an output a new type of context descriptor.

Context model

A mapping from real world concepts to collections of context descriptor states.

Context profile

A context modelling approach which uses collections of key-value pairs, where the key is a context descriptor and the value is a context descriptor state, as groups.

Context scenario definition

Analogous to a context profile but represents collections of raw sensor data rather than context descriptors.

Context

Context is defined as the set of circumstances that describe a collection of external memories. More specifically, context is defined as the set of states of a person and his environment which describe a collection of external memories.

Data channels

Black boxes in a context aware system that both consume and produce data.

Data sinks

Black boxes in a context aware system which consume data.

Data sources

Black boxes in a context aware system, which produce data.

Declarative memory (LTM)

Memories which can be consciously accessed or described by people such as events. Is further divided into Semantic and Episodic memory.

Episodic memory

A system which receives and stores information about temporally dated events and is a record of people's personal lives.

External memories

States of people and their environment as perceived by a machine.

False memory phenomenon

Interpretation and formation of memories of events incorrectly by individuals.

Graph manager

A graph manager is a 2D graph construction tool that enables visual applications (graphs) to be created using black boxes.

Graph model

In the software environment, a graph model represents an application. It consists of a graph (a list of edges and vertices) which is constructed using black boxes.

GUI Views

Components that present a GUI to each of the facilities offered by the software environment.

Intimate computer

A computing device which is part of the personal space of a person, accompanies the person everywhere and processes external memories of the person.

Levels of processing

Processing stimuli from sensors in stages, where each stage represents a different semantic level of interpretation of the stimuli, until features that can be used for encoding or decoding memories are identified.

Lifelog

A record of memories kept by an individual which span long periods of time.

Lifestreams

States of people and their environment prior to their capture by a machine.

Long term memory (LTM)

A subsystem of human memory which stores memories for long periods of time.

Machine based personal memory manager

An entity that consists of multiple artificial sensors and components that analyze sensor data in order to extract and reason on information that can be used to manage an individual's external memories.

Memory

A record of people's experiences and knowledge. (general definition)

Different contexts in which memory is referred to in the thesis:

- Personal memories (a person's memory). Memories which belong to a person.
- A person's biological memory system

- A person's memories as captured by a machine (external memories)
- A piece of hardware

Memory consumers

Entities which retrieve and use memory fragments from the memory manager to support and augment the human memory subsystem or for performing context interpretation.

Memory fragment encoders

Entities which encode raw data sampled by sensors into memory fragments.

Memory fragments

Memory fragments are external memories which are sampled from lifestreams by artificial sensors and digitized for storage on machine-based memory.

Memory manager

A memory manager represents a single logical entity that manages memory fragments by context.

Memory producers

Entities which produce memory fragments and/or context descriptors. Composed of sensors, context interpreters and a context model.

Metadata

Information about data.

Moblog

Weblogs which are maintained using memories captured by mobile PIM devices.

Personal data server

A small portable storage device used to store personal data.

Personal information management (PIM) systems

Systems which provide address book, schedule and communication management features.

Phonological loop

Human memory subsystem used for processing verbal information.

Procedural memory (LTM)

Memories which cannot be consciously accessed or described by people such as skills.

Prospective memories

Memories which describe future events.

Quanta file system layer

Storage medium which is divided into quanta.

Quanta

Plural for quantum.

Quantum

A storage unit which has the properties of a spatial and temporal quantum.

Resource manager

A subsystem of the software environment which facilitates the dynamic loading of components that are part of the software environment at run time.

Retrieval failure

Failure of the human memory system to retrieve memories which exist in storage.

Retrospective memories

Memories about past events.

Semantic memory

People's general knowledge of concepts, rules and languages which make up the world.

Sensor data generation behaviour definition

A definition used to generate sensor data using piecewise functions.

Sensor memory

A human memory subsystem which stores information perceived by the senses for brief periods of time.

Sensors

Entities responsible for sampling raw data from lifestreams.

Short term memory (STM)

A subsystem of human memory which stores memories for short periods of time.

Software environment

An integrated development environment for prototyping context aware applications through a visual programming interface or through a traditional third generation based programming interface.

Spatial quantum

The shortest distance of space supported by a storage system.

State

The value of an attribute of an entity at a particular instance in time

Storage failure

Inability of the human memory system to form new memories due to external influences such as disease, drugs and social situations.

Tagging memories

Labelling memories.

Temporal quantum

A small window into a temporally ordered stream of memory fragments.

Text annotation

The practice of using text supplied by users (text annotation) which contain context descriptors to index people's memories.

Ubiquitous computer

A computing device which stays in the background and provides useful services to an individual without monopolizing user attention.

User model

A model of the states of a user and his environment over periods of time built using context profiles.

Virtual folder

A desktop folder which represents a saved query that can be executed each time a user opens the folder.

Virtual sensors

Sensors emulated by software.

Visio-spatial sketch pad

A human memory subsystem which can process and store visual and spatial information and is essential for learning new visual information.

Wearable computer

A computing device which is incorporated into a person's everyday clothing.

Weblog/Blog

Web based lifelogs.

References

[Ackermann, 2001] Ackermann R, Goertz M, Karsten M and Steinmetz R, "Prototyping a PDA based Communication Appliance", Proceedings of Softcom 2001, Split, Dubrovnik (Croatia); Ancona, Bari (Italy), October 2001.

[Aizawa *et al*, 2001] Aizawa K, Ishijima K and Shiina M, "Summarising Wearable Video", IEEE International Conference on Image Processing (ICIP2001), Thessaloniki (Greece), October 2001.

[Aizawa, 2003] Aizawa K, "Context Based Video Retrieval for Life log Applications", ATR Workshop on Ubiquitous Experience Media (UEM), Kyoto (Japan), September 2003.

[Aizawa, 2004] Aizawa K, "Capture and Efficient Retrieval of Life Log", Pervasive 2004 Workshop on Memory and Sharing of Experiences, Vienna (Austria), April 2004.

[Anderson, 2000] Anderson J.R, *Learning and Memory: An Integrated Approach (2nd Edition)*, Wiley and Sons, New York, 2000.

[Anderson, 2003] Anderson M.C, "Rethinking Interference Theory: Executive Control and the Mechanisms of Forgetting", *Journal of Memory and Language*, Vol. 49, No. 4, pp. 415-445, 2003.

[Aoki *et al*, 1999] Aoki H, Schiele B and Pentland A, "Real time Personal Positioning System for Wearable Computer", Third International Symposium on Wearable Computers (ISWC), San Francisco(USA), October 1999.

[Apple, 2005] Apple, Apple's Spotlight, web ref: http://www.apple.com/macosx/features/spotlight, accessed 10/2005, 2005.

[Asada *et al*, 2003] Asada H, Shaltis P, Reisner A, Rhee S and Hutchinson R, "Mobile Monitoring with Wearable Photoplethysmographic Biosensors", *IEEE Engineering in Medicine and Biology Magazine*, Vol. 22, No. 3, pp. 28-40, 2003.

[ASP.NET, 2005] ASP.NET, Casini .NET Web Server, web ref: http://www.asp.net/Projects/Cassini/Download/Default.aspx?tabindex=0&tabid=1, accessed 10/2005, 2005.

[Atmel, 2003] Atmel, Atmel 8 bit AVR Micro Controller AT90S8535, web ref: http://www.atmel.com/dyn/products/product_card.asp?part_id=2000, accessed 10/2005, 2003.

[Atkinson *et al*, 1968] Atkinson R.C and Shiffrin R.M, "Human Memory: A Proposed System and its Control Processes", *The Psychology of Learning and Motivation*, Vol. 2, pp. 90-95, 1968.

[ATR, 2002] ATR, ATR Media Information Science Laboratorios Interaction Media, web ref: http://www.mis.atr.jp/project/im.html, accessed 10/2005, 2002.

[Autodesk, 2005] Autodesk, 3D Studio Max, web ref: http://www.autodesk.de/adsk/servlet/index?siteID=123112&id=5659302, accessed 10/2005, 2005.

[Baber *et al*, 2002] Baber C and Baumann K, "Embedded Human Computer Interaction", *Applied Ergonomics*, Vol. 33, No. 3, pp. 273-287, 2002.

[Baddeley *et al*, 1974] Baddeley A.D and Hich G.J, "Working Memory", *The Psychology of Learning and Motivation*, Vol. 8, pp. 47-90, 1974.

[Baddeley, 1999] Baddeley A.D, *Essentials of Human Memory*, Psychology Press, East Sussex, 1999.

[Baddeley, 2005] Baddeley G, NMEA GPS Sentence Information, web ref: home.mira.net/~gnb/gps/nmea.html accessed 10/2005, 2005.

[Bardram, 2005] Bardram J.E, "The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications", 3rd International Conference on Pervasive Computing (Pervasive 2005), Munich (Germany), May 2005.

[Bartneck, 2001] Bartneck C, "How Convincing is Mr. Data's Smile: Affective Expressions of Machines", *User Modelling and User-Adapted Interaction*, Vol. 11, No. 4, pp. 279-295, 2001.

[Barton *et al*, 2001] Barton J and Vijayaraghavan V, UBIWISE - A Ubiquitous Wireless Infrastructure Simulation Environment, web ref: http://home.comcast.net/~johnjbarton/ubicomp/ur/ubiwise/index.htm, accessed: 10/2005, 2001.

[Bayons, 2005] Bayon V, Google Desktop .NET API in C#, web ref: http://www.virart.nott.ac.uk/idue/eng/weblog/2005/04/google-desktop-net-api-in-c.html, accessed 10/2005, 2005.

[Beagle, 2005] Beagle, Gnome Desktop Search, web ref: http://beaglewiki.org/Main_Page, accessed 10/2005, 2005.

[Betke *et al*, 2002] Betke M, Gips J and Fleming P, "The Camera Mouse: Visual Tracking of Body Features to Provide Computer Access for People with Severe Disabilities", *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, Vol. 10, No. 1, pp. 1-10, 2002.

[Betke, 2004] Betke M, Video Based Human Computer Interfaces, Video-Based Human-Computer Interfaces, Computer Science Department Boston University, web ref: http://www.cs.bu.edu/faculty/betke/research/hci.html, accessed 10/2005, 2004.

[Beynon-Davies, 1992] Beynon-Davies P, *Relational Database Design*, McGraw Hill, London, 1992.

[Blinkx, 2005] Blinkx Desktop Search, web ref:http://www.blinkx.com, accessed 10/2005, 2005.

[Blogger, 2005] Blogger, Blogger weblog, web ref: http://www.blogger.com, accessed 10/2005, 2005.

[Bristow *et al*, 2004] Bristow H.W, Baber C, Cross J, Knight J.F and Woolley S.I, "Defining and Evaluating Context for Wearable Computing", *International Journal of Human-Computer Studies*, Vol. 60, No. 5-6, pp. 798-819, 2004.

[Brown, 1998] Brown P, "Triggering Information by Context", *Personal Technologies*, Vol. 2, No. 1, 1998.

[Bungert, 2005] Bungert C, Head Mounted Displays/VR-Helmets Market Overview, web ref: http://www.stereo3d.com/hmd.htm#chart, accessed 10/2005, 2005.

[Bush, 1945] Bush V, As We May Think, web ref: http://www.theatlantic.com/doc/194507/bush, accessed 10/2005, 1945.

[Card, 1994] Card R, Introducton to Ex2F, web ref: http://e2fsprogs.sourceforge.net/ext2intro.html, accessed 10/2005, 1994.

[Cardoso, 2004] Cardoso I, C# Mp3 compressor, web ref: http://www.codeproject.com/cs/media/MP3Compressor.asp, accessed 10/2005, 2004

[CARPE, 2004] CARPE, The 2nd ACM Workshop on Capture, Archival and Retrieval of Personal Experiences, web ref: http://research.microsoft.com/CARPE2004, accessed 10/2005, 2004.

[Champion, 2004] Champion J, ZedGraph library, web ref: http://zedgraph.sourceforge.net, accessed 10/2005, 2004.

[Chavez *et al*, 1999] Chavez E, Ide R and Kirste T, "Interactive Applications of Personal Situation-aware Assistants", *Computers and Graphics*, Vol. 23, No. 6, pp. 903-915, 1999.

[Cheng *et al*, 2001] Cheng L and Robinson J, "Personal Contextual Awareness through Visual Focus", *IEEE Intelligent Systems*, Vol. 16., No. 3, pp. 16-20, 2001.

[Cheng *et al*, 2004] Cheng W, Golubchik L, and Kay D, "Total Recall: Are Privacy Changes Inevitable?", First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE), New York (USA), October 2004.

[Christopoulou *et al*, 2004] Christopoulou E, Goumopoulos C, Zaharakis I and Kameas A, "An Ontology-based Conceptual Model for Composing Context-Aware Applications", First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham(England), September 2004.

[Clarkson *et al*, 2000] Clarkson B, Mase K and Pentland A, "Recognizing User's Context from Wearable Sensors: Baseline System", International Symposium on Wearable Computers (ISWC), Atlanta (USA), October 2000.

[Clarkson, 2002] Clarkson B, "Life Patterns: Structure from Wearable Sensors", PhD Thesis, Massachusetts Institute of Technology, 2002.

[CMU Sphinx, 2003] CMU Sphinx, Open Source Speech Recognition engine, web ref: http://www.speech.cs.cmu.edu/sphinx, accessed 10/2005, 2003.

[Copernic, 2005] Copernic, Copernic Desktop Search, web ref: http://www.copernic.com, accessed 10/2005, 2005.

[CoreCodec, 2005] CoreCodec, Core Media Player, web ref:http://tcpmp.corecodec.org, accessed 10/2005, 2005.

[Craik *et al*, 1972] Craik F.I.M and Lockhart R.S, "Levels of Processing: A Framework for Memory Research", *Journal of Verbal and Learning Behaviour*, Vol. 11, pp. 617-684, 1972.

[Cyberkinetics, 2005] Cyberkinetics, Cyberkinetics Neurotechnology Systems, Inc, Cyberkinetics BrainGate, web ref:http://www.cyberkineticsinc.com/content/index.jsp, accessed 10/2005, 2005.

[DARPA Lifelog, 2003] DARPA Lifelog, DARPA Lifelog Program, web ref: http://www.darpa.mil/ipto/programs/lifelog/index.htm, accessed 10/2005, 2003.

[Davis, 1993] Davis M, "Media Streams: An Iconic Visual Language for Video Annotation", IEEE Symposium on Visual Languages, Bergen (Norway), August 1993.

[Davis, 2005] Davis G, Directshow for developers, web ref: http://www.gdcl.co.uk/dshow_dev.htm, accessed 10/2005, 2005.

[Deutscher *et al*, 2004] Deutscher M, Jeffrey P and Siu N, The Memory Collage A Mosaic of Perspectives and Emotions, Paper for UBC Human Interface Technologies, web ref:http://hct.ece.ubc.ca/research/memory, accessed 10/2005, 2004.

[DeVaul *et al,* 2000] DeVaul R and Pentland A, "The Memory Glasses: Towards a Wearable, Context Aware, Situation Appropriate Reminder System", Workshop on Situated Interaction in Ubiquitous Computing at CHI00, Hague (Netherlands), April 2000.

[DeVaul et al, 2003] De Vaul RW and Pentland S, "The Memory Glasses: Subliminal vs Overt memory support with Imperfection", International Symposium on Wearable Computers (ISWC), Florida (USA), October 2003.

[Dey, 2000] Dey A.K, "Providing Architectural Support for Building Context-Aware Applications", PhD thesis, College of Computing, Georgia Institute of Technology, 2000.

[Dey *et al*, 2000b] Dey, A.K and Abowd G.D, "CybreMinder: A Context-Aware System for Supporting Reminders", 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol (UK), September 2000.

[Dey *et al*, 2001] Dey A.K, Salber D and Abowd G.D, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", *Human-Computer Interaction Journal*, Vol. 16, No. 2-4, pp. 97-166, 2001.

[Dey *et al*, 2004] Dey A.K, Hamid R, Beckmann, C, Li I and Hsu D, "a CAPella: Programming by Demonstration of Context-Aware Applications", CHI Conference on Human Factors in Computing Systems, Vienna (Austria), April 2004.

[Doom9, 2005] Doom9, Video Codec Shootout, web ref: http://www.doom9.org/codecs-quali-105-1.htm, accessed 12/2005, 2005.

[Dunlop *et al*, 2002] Dunlop M and Brewster S, "The Challenge of Mobile Devices for Human Computer Interaction", *Personal and Ubiquitous Computing*, Vol.6, No. 4, pp. 235-236, 2002.

[Eagle, 2005] Eagle N, "Machine Perception and Learning of Complex Social Systems", Ph.D. Thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, 2005.

[Eclipse, 2005] Eclipse, Eclipse IDE, web ref:http://www.eclipse.org, accessed 10/2005, 2005.

[Ellis *et al*, 2004] Ellis D and Lee K.S, "Minimal-Impact Audio-Based Personal Archives", First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE), New York (USA), October 2004.

[ExpressPCB, 2004] ExpressPCB, Free PCB Layout Software, web ref: http://www.expresspcb.com, accessed 10/2005, 2004.

[Farringdon *et al*, 1999] Farringdon J, Moore A, Tilbury N, Church J and Biemond P, "Wearable Sensor Badge and Sensor Jacket for Context Awareness", International Symposium on Wearable Computers (ISWC), San Francisco (USA), October 1999.

[Festival, 2003] Festival, The Festival Speech Synthesis System, web ref: http://www.cstr.ed.ac.uk/projects/festival, accessed 10/2005, 2003.

[ffdshow, 2005] ffdshow, A Collection of Open Source Directshow Filters, web ref: http://ffdshow.sourceforge.net, accessed 10/2005, 2005.

[FFmpeg, 2005] FFmpeg, FFmpeg Multimedia System, web ref: http://www.ffmpeg.org, accessed 10/2005, 2005.

[Fitzgibbon *et al*, 2003] Fitzgibbon A and Reiter E, Memories for Life, Managing Information over a Human Lifetime, Grand Challenges for Computing Research Draft Proposals, Sponsored by the UK Computing Research Committee, with Support from EPSRC and NeSC, web ref: http://www.nesc.ac.uk/esi/events/Grand_Challenges/proposals, accessed 10/2005, 2003.

[Flickr, 2005] Flickr, Flickr photo storage, web ref: http://www.flicker.com, accessed 10/2005, 2005.

[Freeman, 1997] Freeman E.T, "The Lifestreams Software Architecture", Ph.D. Dissertation, Yale University Department of Computer Science, 1997.

[Freudenthal *et al*, 2004] Freudenthal A, Hoogh M and Keyson D, "Intelligent Product Builder a rapid prototyping environment for software in context aware hardware products", Proceedings of the conference on Dutch directions in HCI, ACM International Conference Proceeding Series, Vol.65, pp. 3-7, ACM Press, New York, 2004.

[Frigo, 2004] Frigo A, Storing, "Indexing and Retreiving my Autobiography", Pervasive 2004 Workshop on Memory and Sharing of Experiences, Vienna (Austria), April 2004.

[Gasson *et al*, 2004] Gasson M, Hutt H, Goodhew I, Kyberd P and Warwick K, "Invasive Neural Prosthesis for Neural Signal Detection and Nerve Stimulation", *International Journal of Adaptive Control and Signal Processing*, Vol. 19, pp. 365–375, 2004.

[Gilleade *et al*, 2003] Gilleade K, Sheridan J and Allanson J, "Liquid: Designing a Universal Sensor Interface for Ubiquitous Computing", Technical Report (comp-001-2003), Lancaster University, Lancaster, UK, 2003.

[Gemmell *et al*, 2004] Gemmell J, Williams L, Wood K, Lueder R and Bell G, "SenseCam: Passive Capture and Ensuing Issues for a Personal Lifetime Store", First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE), New York (USA), October 2004.

[Gemmell *et al*, 2005] Gemmell J, Aris A, and Lueder R, "Telling Stories with MyLifeBits", IEEE International Conference on Multimedia and Expo (ICME), Amsterdam (The Netherlands), July 2005.

[Glimpse, 2002] Glimpse, Glimpse Index, web ref: http://webglimpse.net, accessed 10/2005, 2002.

[Goldberg *et al*, 1999] Goldberg J and Kotval1 X, "Computer Interface Evaluation using Eye Movements: Methods and Constructs", *International Journal of Industrial Ergonomics*, Vol. 24, No. 6, pp. 631-645, 1999.

[Google, 2005] Google, Google Desktop Search, web ref: http://desktop.google.com, accessed 10/2005, 2005.

[Google, 2005b] Google, Google Web API Reference (beta), web ref: http://www.google.com/apis/reference.html, accessed 10/2005, 2005.

[GNU, 2005] GNU, GNU Find Utilities, web ref: http://www.gnu.org/software/findutils/findutils.html, accessed 10/05, 2005.

[GNU, 2005b] GNU, DotGNU Project, web ref: http://www.dotgnu.org, accessed: 10/2005, 2005.

[GSMA, 2005] GSM Assocation, GSM Facts and Figures, web ref: accessed http://www.gsmworld.com/news/statistics/index.shtml, 10/2005, 2005.

[Gu *et al*, 2005] Gu T, Pung H.K and Zhang D.Q., "A Service Oriented Middleware for Building Context Aware Services", *Journal of Network and Computer Applications*, Vol. 28, No. 1, pp. 1-18, 2005.

[Hagan *et al*, 2002] Hagan R, Zelinsky A and Rougeaux, "Visual Gesture Interfaces for Virtual Environments", *Interacting with Computers*, Vol. 14, No.3, pp. 231-250, 2002.

[Hanson *et al*, 2005] Hanson D, Olney A, Prilliman S, Mathews E, Zielke M, Hammons D, Fernandez R and Stephanou H, "Upending the Uncanny Valley", National Conference on Artificial Intelligence (AAAI), Pittsburgh (USA), July 2005.

[Haya *et a*l, 2004] Haya P. A, Montoro G and Alaman X, "A Prototype of a Context-based Architecture for Intelligent Home Environments", International Conference on Cooperative Information Systems (CoopIS 2004), Larnaca (Cyprus), October 2004.

[Healey *et al*, 1998] Healey J and Picard R.W, "Startlecam: A Cybernetic Wearable Camera", 2nd. International Symposium on Wearable Computers (ISWC), Pittsburgh (USA), October 1998.

[Heckmann, 2003] Heckmann D., "Introducing Situational Statements as an Integrating Data Structure for User Modeling", Workshop on Adaptivity and User Modeling in Interactive Software Systems (ABIS), Germany,October 2003.

[Herrmann *et al*, 1992] Herrmann D.J, Weingartner H, Searleman A and McEvoy C, *Memory Improvement: Implications for Memory Theory*, Springer Verlag, Berlin, 1992.

[Hess, 2003] Hess C, "The Design and Implementation of a Context-Aware File System for Ubiquitous Computing Application", Ph. D Thesis, University Of Illinoisat Urbana-Champaign, 2003.

[Hess *et al*, 2003] Hess C and Campbell R, "A Context-Aware Data Management System for Ubiquitous Computing Applications", International Conference of Distributed Computing Systems (ICDCS 2003), Providence (USA), May 2003.

[Hoffer et al, 1999] Hoffer JA, George JF and Valacich JS, *Modern Systems Analysis and Design (2nd Edition*), Addison-Wesley, Reading, Mass., 1999.

[Hoffman, 1998] Hoffman D, *Visual Intelligence: How We Create What We See*, Norton Publishing, New York, 1998.

[Hoisko, 2003] Hoisko J, "Early Experiences of Visual Memory Prosthesis for Supporting Episodic Memory", *International Journal of Human-Computer Interaction*, Vol. 15, No. 2, pp. 209-230, 2003.

[IAA, 2005] IAA, The Institute of Applied Autonomy, web ref: http://www.appliedautonomy.com/isee.html, accessed 10/2005, 2005.

[IETF, 1987] IETF, Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods, web ref: http://www.ietf.org/rfc/rfc1001.txt, accessed 10/2005, 1987.

[IETF, 1999] IETF, Hypertext Transfer Protocol (HTTP/1.1), web ref: http://www.ietf.org/rfc/rfc2616.txt, accessed 10/2005, 1999.

[IETF, 1999b] IETF, HTTP Extensions for Distributed Authoring (WEBDAV), web ref: http://www.ietf.org/rfc/rfc2518.txt, accessed 10/2005, 1987.

[Intel, 2005] Intel, Open Source Vision Library, web ref: http://www.intel.com/research/mrl/research/opencv, accessed 10/2004, 2005.

[Intille *et al*, 2005] Intille S.S, Larson K, Beaudin J.S, Nawyn J, Tapia E.M and Kaushik P, "A living laboratory for the design and evaluation of ubiquitous computing technologies", Extended Abstracts of the 2005 Conference on Human Factors in Computing Systems, ACM Press, New York, 2005.

[Jameson, 2001] Jameson A, "Modeling Both the Context and the User", *Personal and Ubiquitous Computing*, Vol. 5, No. 1, pp. 29-33, 2001.

[Jay, 2005] Jay G, MeTV, web ref: http://metv.sourceforge.net, accessed 10/2005, 2005.

[Jensen, 2005] Jensen C.S, TDB Glossary: The Consensus Glossary of Temporal Database Concepts, web ref: http://www.cs.auc.dk/~csj/Glossary/index.html, 2005.

[Jonker *et al*, 2003] Jonker P, Persa S, Caarls J, Jong F and Lagendijke I, "Philosophies and Technologies for Ambient Aware Devices in Wearable Computing Grids", *Computer Communications*, Vol. 26, No. 11, pp. 1145-1158, 2003.

[Judd *et al*, 2003] Judd G and Steenkiste P, "Providing Contextual Information to Pervasive Computing Applications", IEEE International Conference on Pervasive Computing (PERCOM), Texas (USA), March 2003.

[Karger *et al*, 2005] Karger D, Bakshi K, Huynh D, Quan D and Sinha V, "Haystack: A General Purpose Information Management Tool for End Users of Semi-structured Data", Second Biennial Conference on Innovative Data Systems Research (CIDR05), USA, January 2005.

[Kawamura *et al*, 2002] Kawamura T, Kono Y and Kidode M, "Wearable Interfaces for a Video Diary: Towards Memory Retrieval, Exchange and Transportation", The 6th IEEE International Symposium on Wearable Computers (ISWC), Seattle (USA), October 2002.

[Kawamura *et al*, 2004] Kawamura T, Amagasa T, Hatano K, Hisazumi K, Kono Y, Murata S, Ueoka T, Ukita N, Miyazaki J, Nakanishi T, Fukuda A, Uemura S and Kidode M, "What is Augmented Memory? - Concept, Design, and Technology", 3rd CREST/ISWC Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing (CREST/ISWC), Virginia (USA), October 2004.

[Kawsar *et al*, 2004] Kawsar F, Fujinami K, and Nakajima T, "Prottoy: A Context Aware Application Framework", 2nd International Symposium on Ubiquitous Computing Systems (UCS), Tokyo (Japan), November 2004.

[KDE, 2005] KDE, K Desktop Environment, web ref: http://www.kde.org, accessed 10/2005, 2005.

[Kern *et al*, 2002] Kern N, Schiele S, Junker H, Lukowicz P, and Tröster G, "Wearable Sensing to Annotate Meeting Recordings", The 6th International Symposium on Wearable Computers (ISWC), Seattle (USA), October 2002.

[Kern *et al*, 2003] Kern N and Schiele B, "Multi-Sensor Activity Context Detection for Wearable Computing", European Symposium on Ambient Intelligence (EUSAI), Eindhoven (The Netherlands), November 2003.

[Kern *et al*, 2004] Kern N, Schiele B, Junker H, Lukowicz P, Tröster G and Schmidt S, "Context Annotation for Lifelong Recording", Pervasive 2004 Workshop on Memory and Sharing of Experiences, Vienna (Austria), April 2004.

[Khedr *et al*, 2005] Khedr M and Karmouch A, "ACAI: Agent-based Context-Aware Infrastructure for Spontaneous Applications", *Journal of Network and Computer Applications*, Vol. 28, No. 1, pp. 19-44, 2005.

[Khungar *et al*, 2004] Khungar S and Riekki J, "Supporting Mobile Context Aware Applications through a Context Aware Storage System", 2nd International Symposium on Ubiquitous Computing Systems, Tokyo(Japan), November 2004.

[Khungar *et al*, 2005] Khungar S and Riekki J, "A Context Cased Storage System for Mobile Computing Applications", *ACM Journal of Mobile Computing and Communications Review*, Vol. 9, No. 1, pp. 64-68, 2005.

[Kim *et al,* 2004] Kim J, Seitz. S and Agrawala M, "Video-Based Document Tracking: Unifying Your Physical and Electronic Desktops", User Interface Software and Technology (UIST), Santa Fe (USA), October 2004.

[Kirillov, 2005] Kirillov A, Motion Detection Algorithms, web ref: http:// http://www.codeproject.com/cs/media/Motion_Detection, accessed 10/2005, 2005.

[Kjeldskov, 2003] Kjeldskov J, "Human-Computer Interaction Design for Emerging Technologies: Virtual Reality, Augmented Reality and Mobile Computer Systems". Aalborg University, Department of Computer Science, Ph.D. Thesis, 2003.

[Koblentz, 2005] Koblentz E, The Evolution of the PDA, web ref: http://www.snarc.net/pda/pda-treatise.htm, accessed 10/2005, 2005.

[Korkea-aho, 2000] Korkea-aho M, Context-Aware Applications Survey, Department of Computer Science, Helsinki University of Technology, web ref: http://www.hut.fi/~mkorkeaa/doc/context-aware.html, accessed 10/2005, 2000.

[Krell, 2003] Krell J, Serial Communications The .NET Way, web ref: http://www.codeproject.com/dotnet/DotNetComPorts.asp, accessed 10/2005, 2003.

[Kwahara *et al*, 2004] Kawahara Y, Hayashi T, Tamura H, Morikawa H and Aoyama T, "A Context-Aware Content Delivery Service Using Off-the-shelf Sensors", The Second International Confernce on Mobile Systems, Applications, and Services (Mobisys 2004 Poster Presentation), Boston (USA), June 2004.

[Laerhoven, 2002] Laerhoven K, The Multi-sensor Wearable, web ref: http://ubicomp.lancs.ac.uk/mat/index.htm, accessed 10/2005, 2002.

[LameEncoder, 2004] LameEncoder, LAME Mp3 Encoder, web ref:http://www.mp3dev.org/mp3, accessed 10/2005, 2004.

[Lamming *et al*, 1994] Lamming M, Brown P, Carter K, Eldridge K, Flynn M, Louie G, Robinson P and Sellen A, "The Design of a Human Memory Prosthesis", *The Computer Journal*, Vol. 37, No. 3, pp. 153-163, 1994.

[Lancos, 2003] Lancos, PonyProg - Serial Device Programmer v2.06c, web ref: http:// http://www.LancOS.com, accessed 10/2005, 2003.

[Leuthardt *et al*, 2004] Leuthardt E.C, Schalk G, Wolpaw J.R, Ojemann J.G and Moran D.W, "A Brain–Computer Interface using Electrocorticographic Signals in Humans", *Journal of Neural Engineering*, Vol. 1, No.2, pp. 63-71, 2004.

[LinuxDevices, 2005] Linux Devices, Tiny SBC's for Embedded Linux Based Projects, web ref:http://www.linuxdevices.com/articles/AT8498487406.html, accessed: 10/2005, 2005.

[Logitech, 2005] Logitech, Logitech pro3000 webcam, web ref: http://www.logitech.com, accessed 10/2005, 2005.

[Low, 2003] Low B, DirectX Capture Wrapper Library, web ref: http://www.codeproject.com/cs/media/directxcapture.asp, accessed 10/2005, 2003.

[LumiSoft, 2003] LumiSoft I, C# SMTP, POP3 and IMAP Mail Server, web ref: http://www.codeproject.com/cs/internet/smtppop3mailserver.asp, accessed 10/2005, 2003.

[Macromedia, 2005] Macromedia, Macromedia Flash, web ref: http://www.macromedia.com, accessed 10/2005, 2005.

[MacIntyre *et al*, 2004] MacIntyre B, Gandy M, Dow S, and Bolter J, "DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences", User Interface Software and Technology (UIST'04), Santa Fe (USA), October 2004.

[Mann, 1997] Mann S, “Wearable Computing: A First Step Toward Personal Imaging”, *IEEE Computer*, Vol. 30, No. 2, pp. 25-32, 1997.

[Mann, 1998] Mann, S, “Wearable Computing as means for Personal Empowerment”, Keynote Address for The first Conference on Wearable Computing (ICWC), Virginia (USA), May 1998.

[Mann, 2001] Steve M, *Cyborg: Digital Destiny and Human Possibility in the Age of the Wearable Computer*, Doubleday Canada, Toronto, 2001.

[Mann, 2004] Mann S, “Continuous Lifelong Capture of Personal Experience with EyeTap”, First ACM Workshop on Continuous Archival and Retrieval of Personal Experiences (CARPE), New York (USA), October 2004.

[McCarty, 1999] McCarty P, Wearable Showcase, web ref: http://wearables.blu.org/showcase, accessed: 10/2005, 1999.

[McDaniel, 2005] McDaniel T, ADO.NET Data Provider for SQLite, web ref: http://sourceforge.net/projects/adodotnetsqlite, accessed 10/2005.

[Mentalis, 2003] Mentalis, .NET security Library, web ref: http://www.mentalis.org/soft/projects/seclib, accessed 10/2005, 2003.

[MicroOptical, 2003] MicroOptical, Miniature Heads Up Displays, web ref: http://www.microopticalcorp.com, accessed 10/2005, 2003.

[Microsoft, 2001] Microsoft, NTFS vs. FAT: Which Is Right for You, web ref: http://www.microsoft.com/windowsxp/using/setup/expert/russel_october01.mspx, accessed 10/2005, 2001.

[Microsoft Speech, 2005] Microsoft Speech SDK, http://www.microsoft.com/speech/download/sdk51, accessed 10/2005, 2005.

[Microsoft, 2005] Microsoft, MSN Desktop Toolbar, web ref: http://desktop.msn.com, accessed 10/2005, 2005.

[Microsoft, 2005b] Microsoft, Microsoft Visual Studio, web ref: http://msdn.microsoft.com/vstudio, accessed 10/2005, 2005.

[Microsoft, 2005c] Microsoft, Directshow, web ref: http://msdn.microsoft.com/library/default.asp?url=/library/enus/directshow/htm/directshow.asp, accessed 10/2005, 2005.

[Microsoft, 2005d] Microsoft, Microsoft Pocket PC Platform, web ref: http://www.microsoft.com/pocketpc, accessed 10/2005, 2005.

[Microsoft, 2005e] Microsoft, Windows XP Remote Desktop, web ref: http://www.microsoft.com/windowsxp/pro/downloads/rdclientdl.asp, accessed 10/2005, 2005.

[Microsoft, 2005g] Microsoft, Windows Vista, web ref http://www.microsoft.com/windowsvista/default.mspx, accessed 10/2005, 2005.

[Mitchell *et al*, 1994] Mitchell T, Caruana T, Freitag D, McDermott J and Zabowski D, "Experience with a Learning Personal Assistant", *Communications of the ACM*, Vol. 37, No.7, pp. 80-91, 1994.

[Mitchell, 2002] Mitchell K, "Supporting the Development of Mobile Context Aware Systems", PhD Thesis, Lancaster University, 2002.

[MIThril, 2003] MIThril, Next Generation Wearable Research Platform, web ref: http://www.media.mit.edu/wearables/mithril, accessed 10/2005, 2003.

[Millard *et al*, 2004] Millard I.C, Roure D and Shadbolt N, "The Use of Ontologies in Contextually Aware Environments", First International Workshop on Advanced Context Modelling, Reasoning and Management in conjuction with UBICOMP04, Nottingham (England), September 2004.

[Minar *et al*, 1999] Minar N, Gray M, Roup O, Krikorian R and Maes P, "Hive: Distributed Agents for Networking Things", Proceedings of the First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents, IEEE Computer Society, Washington DC, 1999.

[mnoGoSearch, 2005] mnoGoSearch, mnoGo Search Engine, web ref: http://search.mnogo.ru, accessed 10/2005, 2005.

[Mobilblogg, 2005] Mobilblogg, Mobilblogg weblog, web ref: http://www.mobilblogg.net, accessed 10/2005, 2005.

[MODI, 2005] MODI, Microsoft Office Document Imaging, web ref: http://www.microsoft.com/downloads/details.aspx?FamilyId=8F93E445-B1CF-4477-A373-E17417D616BC&displaylang=en, accessed 10/2005, 2005.

[Mono, 2005] Mono, Mono.NET, web ref: http://www.mono-project.com/Main_Page, accessed 10/2005, 2005.

[Morris *et al,* 1994] Morris P.E and Gruneberg M.M, *Theoretical Aspects of Memory (2nd Editon)*, Routledge, London, 1994.

[Mplayer, 2005] Mplayer, The Mplayer Media Player, web ref: http://www.mplayerhq.hu, accessed 10/2005, 2005.

[Munoz, 2003] Munoz I, A Full-duplex Audio Player in C# using the WaveIn/WaveOut APIs, web ref: http://www.codeproject.com/cs/media/cswavrec.asp, accessed 10/2005, 2003.

[Narayanaswami *et al*, 2002] Narayanaswami C and Raghunath M.T, "Designing a New Form Factor for Wearable Computing", *IEEE Pervasive Computing*, Vol. 1, No. 4, 2002.

[Nelson, 2002] Nelson R, "Tracking Objects Using Recognition", 16th International Conference of Pattern Recognition, Quebec City (Canada), August 2002.

[NetBeans, 2005] NetBeans, NetBeans Open Source IDE, web ref: http://www.netbeans.org accessed, 10/2005, 2005.

[Netmaster, 2003] Netmaster, Directshow .NET C# Wrapper Library, web ref: http://www.codeproject.com/cs/media/directshownet.asp, accessed 10/2005, 2003.

[Nokia, 2005] Nokia, Nokia Life blog, web ref: http://www.nokia.com/lifeblog, accessed 10/2005, 2005.

[Norman, 1998] Norman D.A, *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*, MIT Press, Cambridge, Mass., 1998.

[OMA, 2005] OMA, Open Mobile Alliance SyncML, web ref: http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html, accessed 10/2005, 2005.

[OpenDX, 2005] OpenDX, IBM Data Explorer, web ref:http://www.opendx.org, accessed 2005, 2005.

[OSAF Chandler, 2005] OSAF Chandler, Open Source Applications Foundation, web ref:http://www.osafoundation.org/Chandler_Compelling_Vision.htm, accessed 10/2005, 2005.

[O'Sullivan, 2005] O'Sullivan D, J2ME Bluetooth Serial Terminal, web ref: http://stage.itp.tsoa.nyu.edu/~dbo3/cgi-bin/ClassWiki.cgi?HelloCellPhone, accessed 10/2005, 2005.

[Palm, 2005] Palm, Palm OS, web ref: http://www.palm.com, accessed 10/2005, 2005.

[Pascoe, 1998] Pascoe J, "Adding Generic Contextual Capabilities to Wearable Computers", 2nd International Symposium on Wearable Computers (ISWC), Pittsburgh (USA), October 1998.

[Parkin, 1997] Parkin A.J, *Memory and Amnesia: An Introduction (2nd Edition)*, Blackwell, Oxford, 1997.

[Parkin, 1999] Parkin A.J, *Memory: A Guide for Professionals*, Wiley and Sons, New York, 1999.

[Patel *et al*, 2004] Patel N and Abowd G.D, "The ContextCam: Automated Point of Capture Video Annotation", The Sixth International Conference on Ubiquitous Computing (UBICOMP), Nottingham (UK), September 2004.

[Pervasive, 2004] Pervasive 04, Second International Conference on Pervasive Computing (PERVASIVE 2004) in Vienna, web ref: http://www.ii.ist.i.kyoto-u.ac.jp/~sumi/pervasive04, accessed 10/2005, 2004.

[Picard *et al*, 1997] Picard R and Healey J, "Affective Wearables", First International Symposium on Wearable Computers (ISWC), Cambridge Mass. (USA), October 1997.

[PlaceLab, 2004] PlaceLab, House_n the PlaceLab, web ref: http://architecture.mit.edu/house_n/placelab.html, accessed 10/2005, 2004.

[Place Lab2, 2005] Place Lab, A Privacy Observant Location System, web ref: http://www.placelab.org accessed, 10/2005, 2005.

[PUJ, 2005] PUJ, Personal and Ubicomp Journal, web ref: http://www.personal-ubicomp.com, accessed 10/2005, 2005.

[PUJ CFP, 2004] Mase K, Sumi Y and Fels S, Call for Papers for the PUJ Special Issue on Memory and Sharing of Experiences, web ref: http://www.ii.ist.i.kyoto-u.ac.jp/~sumi/MSE, accessed 10/2005, 2004.

[Randell *et al*, 2000] Randell C and Muller H. MacIntyre I and Iannucci B, "Context Awareness by Analysing Accelerometer Data", The Fourth International Symposium on Wearable Computers (ISWC), Atlanta (USA), October 2000.

[Rao, 2000] Rao S, Programming the MS Agent Control, web ref: http://www.codeproject.com/tips/msagentctrl.asp, accessed 10/2004, 2000.

[Raskin, 2005] Raskin J, Archy Core Principles, web ref: http://rchi.raskincenter.org/index.php?title=Core_Principles, accessed 10/2005, 2005.

[Rekimoto, 1999] Rekimoto, J, "Time-Machine Computing: A Time-centric Approach for the Information Environment", Proceedings of the 12th annual ACM symposium on User interface software and technology, pp. 45-54, 1999.

[Rekimoto, 1999b] Rekimoto J, "TimeScape: A Time Machine for the Desktop Environment", Conference on Human Factors in Computing Systems (CHI), Pennsylvania (USA), May 1999.

[Rekimoto *et al*, 2000] Rekimoto J and Ayatsuka Y, "CyberCode: Designing Augmented Reality Environments with Visual Tags", Proceedings of DARE 2000 on Designing augmented reality environments, pp. 1-10, 2000.

[Resco, 2003] Resco, Resco Pocket PC Explorer v4.20, web ref: www.resco-net.com, accessed 10/2005, 2003.

[Rhodes *et al*, 1996] Rhodes B and Starner T, "Remembrance Agent: A Continuously Running Automated Information Retrieval System", The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM '96), London (UK), April 1996.

[Rhodes, 1997] Rhodes B, A Brief History of Wearable Computing, web ref: http://bradleyrhodes.com/Papers/brief-history-of-wearable-computing.html, accessed 10/2005, 1997.

[Rhodes, 1997b] Rhodes B, "The Wearable Remembrance Agent: A System for Augmented Memory", First International Symposium on Wearable Computers (ISWC), Cambridge Mass. (USA), October 1997.

[Rhodes, 2003] Rhodes J, "Using Physical Context for Just-in-Time Information Retrieval", *IEEE Transactions on Computers*, Vol. 52, No. 8, pp. 1011-101, 2003.

[Riva, 2004] Riva O, "A Conceptual Model for Structuring Context-Aware Applications", Fourth Berkeley - Helsinki Ph.D. Student Workshop on Telecommunication Software Architectures, University of Berkeley, 2004.

[Rohs *et al*, 2004] Rohs M and Gfeller B, "Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects", Advances in Pervasive Computing, Austrian Computer Society (OCG), Vienna (Austria), April 2004.

[Salber *et al*, 1999] Salber D, Dey A and Abowd A, "The Context Toolkit: Aiding the Development of Context Enabled Applications", Conference on Human Factors in Computing Systems (CHI99), Pennsylvania (USA), May 1999.

[SemaCode Corp, 2005] SemaCode Corp, SemaCode, web ref: http://SemaCode.org, accessed 10/2005, 2005.

[Sanner *et al*, 2002] Sanner M.F, Stoffler D and Olson A.J, ViPER, "A Visual Programming Environment for Python", The 10th International Python Conference, Virginia (USA), February 2002.

[Schilit *et al*, 1994] Schilit B.N, Adams N and Want R, "Context-Aware Computing Applications", IEEE Workshop on Mobile Computing and Applications, California (USA), December 1994.

[Schilit, 1995] Schilit W, "Context Aware Mobile Computing", PhD Thesis, Columbia University, 1995.

[Schilit *et al*, 2003] Schilit B, LaMarca A, Borriello G, Griswold W, McDonald D, Lazowska E, Balachandran A, Hong J and Iverson V, "Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative", The First ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH), San Diego (USA), September 2003.

[Schmidt *et al*, 1999] Schmidt A, Aidoo K, Takaluoma A, Tuomela U, Laerhoven K and Velde W, "Advanced Interaction in Context", Handheld and Ubiquitous Computing First International Symposium (HUC), Karlsruhe (Germany), September 1999.

[Schmidt *et al*, 2001] Schmidt A and Gellersen T, "Context-Phonebook Extending Mobile Phone Applications with Context", Third International Workshop on Human Computer Interaction with Mobile Devices, Lille (France), September 2001.

[Schmidt, 2002] Schmidt A, "Ubiquitous Computing - Computing in Context", PhD Thesis, Lancaster University, 2002.

[Schraefel *et al*, 2005] Schraefel M. C, Smith, D. A, Owens A, Russell A, Harris C and Wilson, M. L, "The evolving mSpace platform: leveraging the Semantic Web on the Trail of the Memex", Sixteenth ACM Conference on Hypertext and Hypermedia, Salzburg (Austria), September 2005.

[Schutte, 2005] Schutte K, Slogger Firefox Extension, web ref:http://slogger.mozilla.org, accessed 10/2005, 2005.

[Scott *et al*, 2005] Scott J and Dragovic B, "Audio Location Accurate Low-Cost Location Sensing", 3rd International Conference on Pervasive Computing (Pervasive 2005), Munich (Germany), May 2005.

[Sebe *et al*, 2005] Sebe N, Cohen I and Huang T.S, *Handbook of Pattern Recognition and Computer Vision*, World Scientific, London, 2005.

[Senseable.MIT, 2005] Senseable.MIT, MIT Senseable City Lab, web ref: http://senseable.mit.edu/news/on_us/CNN4November2005.htm, accessed 11/2005, 2005.

[Simpson *et al*, 1996] Simpson R, Renear A, Mylonas E and Dam A, "50 years after "As we may think": the Brown/MIT Vannevar Bush symposium", *ACM Interactions*, Vol. 3, No.2, pp. 47-67, 1996.

[Small *et al*, 2000] Small J, Smailagic A and Siewiorek D, Determining User Location for Context Aware Computing Through the Use of a Wireless LAN Infrastructure, web ref: http://www.cs.cmu.edu/~aura/docdir/small00.pdf, accessed 10/2005, 2000.

[SocketCom, 2005] SocketCom, Socket GPS Receiver, web ref: http://www.socketcom.com/product/GP0820-521.asp, accessed 10/2005, 2005.

[SocketCom, 2005b] Socket WiFi card, web ref: http://www.socketcom.com/product/WL6004-322.asp, accessed 10/2005, 2005.

[SQLite, 2005] SQLite, Portable SQL Database, web ref: http://www.sqlite.org, accessed 10/2005, 2005.

[Stang, 2002] Stang P, Analog to Digital Converter Functions, web ref: http://ccrma-www.stanford.edu/courses/250a/docs/avrlib-new/a2d_8c.html, accessed 10/2005, 2005.

[Starner, 1999] Starner T, "Wearable Computing and Contextual Awareness", PhD Thesis, MIT Media Laboratory, 1999.

[Stauffer *et al*, 2000] Stauffer C, Eric W and Grimson L, "Learning Patterns of Activity Using Real-Time", Tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 8, pp. 747-757, 2000.

[Stanford, 2003] Stanford V, "Pervasive Computing Goes the Last Hundred Feet with RFID Systems", *IEEE Pervasive Computing*, Vol. 02, No. 2, pp. 9-14, 2003.

[Strang *et al*, 2004] Strang T and Linnhoff-Popien C, "A Context Modelling Survey", First International Workshop on Advanced Context Modelling, Reasoning and Management part of part of UBICOMP 2004, Nottingham(England), September 2004.

[Sumi *et al*, 2004], Sumi K, Sugimoto A, Matsuyama T, Toda N and Tsukizawa S, "Active Wearable Vision Sensor Recognition of Human Activities and Environments", Proceedings of the International Conference on Informatics Research for Development of Knowledge Society Infrastructure (ICKS), pp. 15-22, IEEE Computer Society, Washington DC, 2004.

[Sun, 2005] Sun, J2ME Platform, web ref: http://java.sun.com/j2me, accessed 10/2005, 2005.

[Symbian, 2005] Symbian, Symbian OS, web ref: http://www.symbian.com, accessed 10/2005, 2005.

[TekGear, 2005] TekGear, TekGear Inc., web ref: http://www.tekgear.ca, accessed 10/2005, 2005.

[Teller, 2004] Teller A, "A Platform for Wearable Physiological Computing", *Interacting With Computers*, Vol. 16, No.5, pp. 917-937, 2004.

[TextAmerica, 2005] TextAmerica, Text America weblog, web ref: http://www.textamerica.com, accessed 10/2005, 2005.

[TinyCad, 2004] TinyCAD, Open Source Schematic Capture Programme for Windows, web ref: http://www.expresspcb.com, accessed 10/2005, 2004.

[TrueCrypt, 2005] TrueCrypt, TrueCrypt Encryption System, web ref: http://www.truecrypt.org, accessed 10/2005, 2005.

[Tsegaye *et al*, 2004] Tsegaye M, Bangay S and Terzoli A, "Using Audio to Enhance Personal Information Management", Southern African Telecommunication Networks and Applications Conference(SATNAC), Stellenbosch (South Africa), September 2004.

[Tsegaye *et al*, 2004b] Tsegaye M, Bangay S and Terzoli A, Wearing your PIM: Experiments with an Audio Enhanced PIM (TR2), web ref: http://www.cs.ru.ac.za/research/students/g98t4414/research accessed 10/2005, 2004.

[Tsegaye *et al*, 2005] Tsegaye M, Bangay S, Terzoli A, "A Prototyping Environment for Investigating Context Aware Wearable Applications", Southern African Telecommunication Networks and Applications Conference(SATNAC), Drakensberg (South Africa), September 2005.

[Toko, 1998] Toko K, "Electronic Tongue", *Biosensors and Bioelectronics*, Vol. 13, No. 6, pp. 701-709, 1998.

[Tulving, 1985] Tulving E, *Elements of Episodic Memory*, Oxford University Press, USA, 1985.

[Ueoka *et al*, 2004] Ueoka T, Kawamura T, Baba S, Yoshimura S, Kono Y and Kidode M, "Camera Device for Supporting Object-Triggered Memory Augmentation", 3rd CREST and ISWC Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing, Virginia (USA), October 2004.

[UK Challenge, 2005] UK Challenge, Memories for Life Grand Challenge, web ref: http://www.memoriesforlife.org, accessed 10/2005, 2005.

[UPnP, 2005] UPnP Forum, UPnP Forum, web ref:http://www.upnp.org, accessed 10/2005, 2005.

[Vemuri *et al*, 2004] Vemuri S and Bender W, "Next-Generation Personal Memory Aids", *BT Technology Journal*, Vol. 22, No. 4, 2004.

[VideoLan, 2005] VideoLan, VLC media player, web ref: http://www.videolan.org, accessed 10/2005, 2005.

[W3C, 2001] W3C, The Semantic Web, web ref: http://www.w3.org/2001/sw, accessed 10/2005, 2005.

[Want *et al*, 2000] Want R and Russell D, "Ubiquitous Electronic Tagging", *IEEE Distributed Systems Online*, Vol. 1, No. 2, 2000.

[Want *et al*, 2002] Want R, Pering T, Danneels G, Kumar M, Sundar M and Light J, "The Personal Server: Changing the Way we Think about Ubiquitous Computing", 4th

International Conference on Ubiquitous Computing, Goteborg (Sweden), September 2002.

[Warwick, 2005] Warwick K, What happens when a man is merged with a computer?, web ref: http://www.kevinwarwick.com/Cyborg1.htm, accessed 10/2005, 2005.

[Weiser, 1991] Weiser M, "The Computer for the 21st Century", *Scientific American*, Vol. 265, No. 3., 1991.

[Weiser, 1994] Weiser M, "The World is not a Desktop", *ACM Transactions*, Vol 1., No 1., pp. 7-8, 1994.

[Welter, 2005] Welter P, The NetSpell spell checker for .NET, web ref: http://www.loresoft.com/NetSpell, web ref accessed 10/2005, 2005.

[Wentworth *et al*, 2003] Wentworth P and Zhao X, Open CV C# Wrapper, web ref: http://www.cs.ru.ac.za/research/sharpercv, accessed 10/2005, 2003.

[Wilson, 2005] Wilson J, *Sensor Technology Handbook*, Elsevier, Oxford, 2005.

[WinAVR, 2004] WinAVR, AVR GCC for Windows, web ref: http://winavr.sourceforge.net, accessed 10/2005, 2004.

[Winograd, 2001] Winograd T, "Architectures for Context, Human-Computer Interaction", *Special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal*, Vol. 16, No. 2-4, pp. 97-166, 2001.

[WordNet, 2005] WordNet, WordNet: A lexical database for the English language, web ref: http://wordnet.princeton.edu, accessed 10/2005, 2005.

[Wu, 2006] Wu T, Keeping Secrets: A Simple Prescription for Keeping Google's Records out of Government Hands, web ref:http://www.slate.com/id/2134670/?nav=tap3, accessed 30/2006, 2006.

[Wyszynski *et al*, 2005] Wyszynskia B, Yamanakab T and Nakamotob T, Recording and reproducing citrus flavors using odor recorder, *Sensors and Actuators B: Chemical*, Vol. 106, No.1, pp. 388-393, 2005.

[XviD, 2005] XviD, Xvid MPEG 4 Video Codec, web ref: http://www.xvid.org, accessed: 10/2004, 2005.

[York *et al*, 2004] York J and Pendharkar P, "Human–Computer Interaction Issues for Mobile Computing in a Variable Work Context", *International Journal of Human Computer Studies*, Vol. 60, No. 5-6, pp. 771-797, 2004.

[Zhai *et al*, 2005] Zhai S, Kristensson P and Smith B, "In Search of Effective Text Input Interfaces for Off the Desktop Computing", *Interacting with Computers*, Vol. 17, No. 3, pp. 229-250, 2005.