CONTEXT-AWARE SOFTWARE

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT AT CANTERBURY

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

By Jason Pascoe

August 2001

Acknowledgements

There are many people to thank for their invaluable help in completing this thesis, but first I would like to acknowledge the financial help of the University of Kent at Canterbury's E.B.Spratt Award, without which this thesis would never have got started, and the JTAP (JISC Technology Applications Programme) funding for the "Mobile Computing in a Fieldwork Environment" project, under which much of the practical work of this thesis was carried out. Additionally, the ideas for distribution described in the final part of the thesis were largely developed under a six-week short-term research fellowship at BT Labs. I am also grateful to the University of Kent's Computing Laboratory, which among many other things helped fund a number of trips.

My supervisor, Peter Brown, has been a great source of help, support, and humour during the whole PhD process, and it was his original ideas that inspired the work herein. I suspect the time required to complete this PhD was somewhat longer than Peter had originally envisioned, as it has lasted well into his retirement years. So I must particularly thank him for stubbornly sticking with this last PhD student of his during an otherwise tranquil retirement in Devon. Of course, it would also not do to let this opportunity pass without acknowledging Peter's sense of humour which, although often at my expense, made for a lively and fun PhD. A good example being the time he convinced me to attend an otherwise suit-and-tie-clad conference in England's House of Commons whilst dressed up in my African safari regalia! I certainly made an impression…

This brings me nicely on to thank my office partners Geraldina, Helena, and Vince who helped convince Peter that I was profitably engaged on the seldom occasions I was found absent from the office. Thanks for your tireless excuses! It was a pleasure to share an office and become great friends with the three of you. Many good times were had in and out of the office including trips to sci-fi movies (generally at my insistence!) and Portuguese salty seafood experiences (generally at their insistence!), and some completely odd experiences such as discovering in one corner of the office a box full of Japanese Elvis-impersonator music tapes and, stranger still, a container full of labelled rhinoceros faeces! (One can only wonder about the previous office occupants…). Fun

I would like to extend a huge "thankyou" to my family for their love and support during this whole process. Thanks to my parents for trusting that it was the right decision for me to do the PhD, supporting me throughout it in every sense, and urging me on to the finish. I suspect that during the write-up you wondered if it would ever be completed. Well, I'm happy to have successfully finished and to tell you that without you it would never have even got started. Mum and Dad, thanks for everything!

Finally, I would like to especially thank my grandparents for simply being wonderful. I was incredibly lucky to have started out with such good people, funny stories, and lovely walks in the countryside. My sense of all that's good in the world comes from you. Gran and Grandad, you're the best and are always in my heart.

To end the acknowledgements on a somewhat humorous note I should also thank Morani, pictured with me below, whose somewhat surprising appearance (for a computing project's web page in any case) has undoubtedly been a great help in our efforts to publicise and spread word of our work.

Contents

Table of Figures

Table of Tables

Chapter 1:

Introduction

Computers are moving ever closer to and becoming ever more entwined with us and our everyday world. The first large mainframe computers were locked away in central processing rooms, never to be seen by most users. Punched cards and printouts formed the limited connection between human and computer. This connection has since been greatly strengthened, firstly with dumb terminals, followed by more intelligent terminal equipment, and culminating in today's predominant manifestation of computing: the desktop personal computer. Of course, the richness of human-computer-interaction (HCI) has also greatly improved, with communication today most commonly conducted at a very abstract level through sophisticated graphical user interfaces (GUIs).

The desktop is by no means the final destination of the computer. Laptop computers already permit users to bring their "desktop" with them anywhere. Perhaps more interesting is the increasing proliferation of personal digital assistants (PDAs) or palmtop computers (we use the two terms synonymously). These devices, most simply stated, are electronic counterparts to paper-based agendas, diaries, notepads and day planners. However, with increasing miniaturisation, component cost reduction, and integration of wireless communications technology, they are being transforming into more sophisticated devices that can offer services such as music, video and continuous Internet access. Because they are small and light the computer is freed from the user's desktop or lap and is beginning to be used anywhere and at anytime, moving further into our everyday world.

With this accelerating penetration of computers into everyday locations we will also start to see an increased integration of these devices with their environment and users, and not just in the palms of their hands. Wearable computing researchers [Starner, Rhodes 1999 - #135] suggest that users will find their computers so indispensable in

their everyday lives that they will come to wear them, in a similar way as to how people have come to wear time pieces on their wrists. Another area of research, that of ubiquitous or pervasive computing [Weiser 1991 - #149], suggests that computing technology will become so specialised and well-integrated into our physical world that we will no longer be aware of it in itself, just as we would now not particularly think of the pen or pencil technology that we use when writing some notes on a sheet of paper.

Simply building the physical palmtop, wearable, or ubiquitous computing devices is not enough to ensure their success. New applications, and the enabling technologies to support them, will be required in order to induce users to invest in this new form of computing. For example, with the advent of desktop computers users had a relatively high performance and fast feedback computer system. However, they did not invest in these computers for technology's own sake, but rather for the work they could do with applications such as spreadsheets and word processors, which were supported with enabling technologies such as graphical user interfaces, interaction devices, storage media, etc.

We believe that context-awareness will be a key enabling technology in the world of palmtop, wearable, and ubiquitous computing, and that most applications running on these devices will be, to some degree, context-aware software. Rather than trying to offer the same service irrespective of the user's current context (e.g. location, time, task at hand, etc.) we believe that software will try to understand that context and use it to react and adapt in order to offer a better or more appropriate service to the user. This need stems from the fact that the computer is no longer shackled to a single desktop but is now part of the user's ever-changing world. This also impacts the device's user interface, which cannot afford to follow the same uniform rules of desktop computing but must instead employ new strategies that better mould it to the dynamic context of use.

1.1   Motivation and Aims of this Thesis

Context-awareness is a new discipline, and we are still a long way from finding and understanding the best approaches. This thesis aims to make a contribution by proposing, and in most cases implementing and testing for usability, a set of new

techniques and approaches. More specifically we aim to further the state-of-the-art in context-aware software on three levels:

Level 1: Infrastructure

> *Research Aim:* to investigate if software infrastructure to support context-awareness is required, and if so, to determine what services that infrastructure should provide and how it should provide them.

> *Work Done:* as a first step, in chapter 2 we present a framework for building context-aware applications based on the application-level metaphor of electronic Post-It notes. Some applications were built using this framework and in chapter 6 we re-examine the framework, and other frameworks proposed by other researchers, and assess what is and what is not genuinely useful in an infrastructure to support context-awareness. As a result of this analysis in the subsequent two chapters we propose a lower-level service that will support a broader range of context-aware software.

Level 2: Applications

> *Research Aim:* to further the understanding of context and context-awareness through empirical experiments, to explore practical applications of context-awareness, and to find out if context-awareness can be usefully employed in practice.

> *Work Done:* in chapter 3 we explore one particular application of context-aware software for palmtop computing, one with a particularly dynamic environment: that of in-field data collection and observation in an African game reserve.

Level 3: HCI

> *Research Aim:* to explore new methods of human-computer interaction for this new area of computing, to determine their usability, and to establish some general HCI design principles.

> *Work Done:* in chapter 4 we present some issues to consider when designing user interfaces for these new breeds of computing device, and provide a

method of mapping the design space for a specific environment, user, and device/application set. Chapter 5 continues our HCI investigation with a usability study conducted on the users of the fieldwork applications we developed in chapter 3.

1.2   Background

We present here a background and survey of the other work related to our research on context-aware software. This overview of research aims to set the scene for the broad array of new emerging technologies in which context-awareness will play an important factor. We divide the survey into three sections: physical technology, applications, and software infrastructure.

1.2.1   Physical Technology

There are three pronounced trends in technology of most concern to us:

    i.   Increasingly small and fast computing devices, enabling computers to be used where needed and not just on the desktop

    ii.   Better sensor technologies, enabling computers to find out more about the new physical environments in which they find themselves.

    iii.   Better communications technologies, allowing computers to maintain access to networked resources whilst away from the desktop.

1.2.1.1   New Computing Devices

The ParcTAB project at Xerox Parc [Want, Shilit 1995 - #146], which used earlier experience with the location technology of the active badge (see later), was one of the first ventures into developing and deploying a new type of computing device. The work aimed to develop a device employees would want to carry with them and use anywhere within the XeroxPARC laboratories. Unlike a laptop this device would be *really* portable and emphasise the importance of context and communications in the applications that it offered. Communications in fact were critical as the device was little more than a

dumb graphic terminal, with its intelligence located in a network-side proxy called the TAB agent. Communications were implemented via an Infrared network that was comprised of room-sized cells, allowing a central server to use the communications infrastructure to monitor the location of devices, and hence users, too.

The physical TAB device was a radical rethink of computer ergonomics and provided a small, light and aesthetically pleasing device that encouraged user acceptance. The user interface provided a touch screen, pen interface (using a combination of Goldberg's Uni-strokes [Want, Shilit 1995 - #146] and onscreen keyboard) and three physical push-buttons.

Success was defined on an application-by-application basis, where some applications proved more popular than others. Generally speaking though, the device was very popular, although the need for a form of disconnected operation was highlighted, useful in times when the user is out of range of the wireless network.

Although a dead-end in itself the TAB device can be seen as a precursor to many of the commercially available PDAs and palmtop computers available on the market today. The Apple Newton broke a lot of new ground in using similar ideas to the TAB and turning them into a viable product concept. Many still regard it as superior in many ways to its successors, but it was ultimately let down by its relatively large form factor, sluggish responsiveness, and poor hand-writing recognition in its early models.

Fadel [Fadel 1995 - #38] argued that any truly useful PDA must be able to communicate with LANs, WANs, peripherals, and other PDAs using wireless links. The concept of directly interfacing a PDA to a kind of "mother ship" desktop computer to share files was also proposed. Similarly Meth [Meth 1994 - #89] puts forward connectivity and communications as key ingredients to a PDA, along with good user interfaces and software applications.

Palm Computing [Palm 2001 - #96] took these lessons, along with its own insight that the simplicity of the user interface and applications is key, and developed the Palm range of PDA devices. The combination of simplicity, very small form factor, and limited connectivity (via a serial cable to the PC, and via IrDA to other Palm devices)

proved a winning combination that now sees Palm as the dominant player in this market.

Today's market is by no means the end of the trail: quite the opposite in fact. A large focus of interest today is in the convergence of mobile phone and PDA technology into what has been termed by the industry as smart phones or personal communicators. Smith [Smith 1995 - #133] writes of a Personal Intelligent Communicator (PIC) that combines voice and data communications, and also combines intelligence in the device-based application software and network-based service software. The philosophy of these devices is more on communications with computing than computing with communications. The technology to build these devices is essentially here, and companies such as Symbian [Symbian 2001 - #139] are about to capitalise on this market.

Personal communicators can be seen as the next logical progression from today's PDAs and mobile phones. However, on a somewhat different branch of computer evolution are wearable and ubiquitous computers. Instead of a device that the user may choose to take with them, wearable computers, such as those being developed at Carnegie Mellon University [Smailagic and Siewiorek 1996 - #132] [Finger, Terk 1996 - #45], are more intimately associated with the user, either embedded in clothing or worn similar in fashion to a wrist-watch. The ultimate integration of a wearable computer could physically link flesh and silicon. However, the more modest aim of wearables right now is to continuously provide computing resources to the user wherever she is, and whatever she is doing. An analogy is often drawn with clocks in order to demonstrate how a technology can become so indispensable that we want wear it, i.e. the wrist-watch. As Selker points out [Selker 1996 - #130], wearables are already starting to be used in industry but there are many user interface issues to be resolved, and not just in the direct interaction of human with machine but also in the larger social picture. To continue the wrist-watch analogy, a number of social statements are inferred from one's watch, not least if looking at it whilst in a conversation one can assume the person is bored or is eager to leave [Feiner 1999 - #40]. Many more situations similar to these are likely to confront wearable users.

The final computing device technology we wish to address is that of ubiquitous computing, a term coined by Mark Weiser [Weiser 1991 - #149, Weiser 1998 - #150].

The term pervasive computing is also often used synonymously. The ubiquitous computing concept lies on the other end of the spectrum to wearable computing, proposing that rather than users possessing their own personal devices they will find the computing equipment they desire wherever they need it in their environment and in all shapes and sizes to suit the task at hand. There is an implicit need for an advanced wireless communications infrastructure so that the electronic resources the user wishes to work with are accessible from the device they choose to use. Transparent ease of use is also another essential requirement to ubiquitous computing: the devices themselves should be transparent to the task at hand much like how a person drives a car to a destination without thinking about the technology at hand or how it works. Perhaps the most difficult requirement to satisfy in the immediate future is the need for almost disposable pricing levels of computers in order for them to be freely distributed about our environment. However, a creeping proliferation could already be conceived to have started with computing equipment such as PCs, set-top boxes, PDAs, and new devices such as web-enabled phones and microwaves, advancing into our homes and everyday world.

1.2.1.2 *New Sensors*

Whether it be palmtop, communicator, wearable, or ubiquitous, new types of computing device are becoming more and more present and integrated into our everyday lives. In so doing these devices are exposed to dynamic ever-changing environments of use, with completely different parameters than that of the world of the desktop. No longer can the computers expect the user to work within its *artificial* desktop world, the computer must now work within the user's *real* world. And to do this effectively they need to become context-aware, spurring the need for some ability to sense the physical world around them, most notably their location.

One of the first and most well known forays into location sensing was the Active Badge Network developed at AT&T Labs (formerly Olivetti Research Labs) [Bennet and Harper 1993 - #8] [Want, Hopper 1992 - #145]. Primarily intended for indoor use, this system required the installation of at least one Infrared transceiver per room, which was physically connected to a networked "condenser" that forwarded location sightings to a centralised server. People or equipment whose location should be monitored were

issued with a small badge that identified the person/equipment with a unique ID. This ID would be transmitted over Infrared and detected by the nearest transceiver that in turn reported it to a central server, allowing it to maintain a model of the current whereabouts of people and equipment.

Although technically very successful the active badge network faced a number of problems. Firstly, it required an initial non-trivial installation of transceivers that required lots of physical wiring to the communications and power network. Secondly, being a centralised system, user's are justifiably concerned about the security and privacy of their location information and what it is being used for. Thirdly, the room-sized location granularity is fine for some applications, such as the automatic phone call routing prototype developed, but higher granularity is needed for more sophisticated applications. Fourthly, it requires users to attach an electronic badge to their clothing, which would often be forgotten (sometimes deliberately) or the garment it was attached to removed, etc.

The Active Floor [Addlesee 1997 - #3] attempted to resolve the latter problem through specially equipped floor tiles that would recognise a user through nothing other than their footsteps. Furthermore, the floor tiles could also try to gauge a user's mood or the items they were carrying through variations in the footstep pattern. Unfortunately, although providing a better granularity of location data this system requires even more infrastructure investment than with active badges, the privacy issue remains unresolved, and there are huge technical challenges in developing the footstep recognition algorithms.

The most recent reincarnation of Active Badges are Active BATs [Ward, Jones 1997 - #147]. Similar in general principle, this system outfits the ceiling of a room with an array of *ultrasound* transceivers that are used in collaboration to triangulate the position of a user within the room, based on the travel time of a ping from her BAT to each of the listening transceivers. This system greatly improves the location accuracy of active badges to a level of centimetres, allowing for a much broader range of applications. Furthermore, this level of location granularity permits two BATs per person/equipment to be used in order to determine orientation in addition to location. The level of accuracy of the system makes it quite appealing for indoor applications despite the infrastructure costs, although the location privacy issue remains unresolved.

The Cricket [Priyantha, Chakraborty 2000 - #107] and Locust Swarm [Kirch, Starner 1997 - #72] systems developed at MIT overcome both infrastructure cost and privacy concerns with a non-networked distributed solution. Small devices, called Locusts, are suspended from below the lighting in rooms and take their energy from solar cells mounted on the top of the device. A person's PDA may communicate with a Locust in its immediate vicinity through IrDA or radio signals (both communications mediums have been tried in two different systems). The locust device is not very intelligent, only allowing clients to set and query a limited number of name-value pairs that it maintains. These name-value pairs can be used to store information about the environment, such as the location, owner of the room, etc. Although the locusts are completely disconnected to anything other than the clients that approach them, it has been proposed that they could be used for relaying messages by piggybacking them onto client devices that transfer them to the next locust, similarly in principle to how a virus spreads.

One technical problem remains with the locust devices, and that is in how they communicate with clients. If using IrDA then the effective communication range of most PDA devices is only 2-3 feet, requiring either a high density of locusts or for the user to closely approach them – neither option is particularly attractive. Using radio-based communications solves this problem but this requires adding special radio transceiver equipment to existing PDA devices – another unattractive prospect. Both problems also face passive radio tag technology, which requires special detection equipment (that sends out a pulse that is "reflected" and augmented by the tag to include its unique ID) and close proximity of the detecting device in order to function.

IrDA beacons developed at lesswire AG [lesswire 2001 - #82] take their inspiration from locusts but are more similar in operation to a lighthouse, in that they transmit a pre-programmed beacon identifier at regular intervals (typically 1-2 seconds). The beacons use high power Infrared LEDs allowing beacon transmissions of up to 8 metres with an angular spread of 45 degrees. As the beacons only broadcast information (i.e. no return communication is required from the client) any of today's PDA devices may use the beacons, regardless of their return IrDA signal strength, and many PDAs may receive the same beacon transmission simultaneously. The PDA may

contain a mapping table of beacon IDs to locations or it may *choose* to report its location to a central server in a similar fashion to active badges.

One form of unique ID system that is already widely deployed today, though typically for object rather than location identification, is the common barcode. However, although barcodes are supremely inexpensive compared to IrDA beacons or any other electronic form of tagging, a computing device must be equipped with a relatively expensive laser scanner in order to detect them, and must typically be closer than one metre to perform a scan successfully. Although suited to specialist applications, such as inventory management, they do not make for a mass market and transparent location sensing system. The CyberCode system developed by Rekimoto [Rekimoto and Ayatsuka 2000 - #110] aims to address and improve upon some of these deficiencies with more sophisticated forms of barcode that can be recognised by digital cameras and successfully operated at much greater distances from the target barcode. Although sharing the extremely low infrastructure cost of normal barcodes (CyberCodes can be easily prepared, printed, and stuck to a wall, door, or any other object) they do require the detecting computing device to be equipped with a digital camera. This requirement may not be so excessive in future computing devices, especially wearables, which often incorporate digital camera apparatus. In fact, some research in analysing the video images from digital cameras [Janin, Zikan 1994 - #68] is exploring how to recognise the environment in a similar fashion to the way we humans do. However, this is an extremely difficult task and is not likely to be viable in the near future.

The sensor solutions discussed thus far are primarily intended for indoor areas as they require the environment to be outfitted with special equipment. Some of these sensors may also be used in outdoor environments although they are still limited, for obvious practical reasons, to confined areas.

By far the most common location-finding technology in outdoor environments is the Global Positioning System (GPS) [Texas-University 1999 - #141] [Herring 1996 - #59], which is now in common use by everyday consumers for activities such as route finding on holiday [Wright 1995 - #153]. Twenty-four satellites orbiting the earth ensure that a ground-based receiver with a clear line of sight will be able to "see" at least four satellites from which it may compute its own position (using the communication time period, and hence calculable distance, to each satellite as the basis). Initially the accuracy

of the GPS signal was deliberately downgraded by the military of the USA (who own the GPS satellite system) with an introduced error system, called *selective availability*. Only military receivers were able to decode the GPS signal without error. However, except in times and places of conflict, selective availability has now been removed and common civilian receivers can obtain location accuracies within 2-10 metres. With planet-wide coverage, relatively good accuracy, and no infrastructure or usage fee (just the cost of the receiver), GPS has become the dominant outdoor positioning technology. Unfortunately the signals used cannot penetrate buildings or even dense vegetation, preventing their use indoors and causing some problems in urban canyons such as skyscraper lined city streets.

Of course, sensor technology is not limited to the detection of location, although location does tend to receive most attention in context-aware computing circles due to the obvious relevance for computing devices that are increasingly mobile. Sensing location is also considerably more complex when compared to sensing light levels, temperature, movement, magnetism, time, etc. For the latter senses, standard technologies exist which can be re-used and adapted from other industries, but location sensing still poses many research issues.

Although well understood and in existence for some time, conventional sensors may also be deployed in new and innovative ways. Harrison et al [Harrison, Fishkin 1998 - #57] equipped standard PDA devices with various tilt and touch sensors that allowed physical movements of the device to be used as a means of interaction. For example, a list of items could be scrolled down by physically tilting the device downwards, and vice versa to scroll the list up.

Radically new types of sensor may also have yet to be devised. Rudall [Rudall - #118] describes a system comprised of a row of electronic sensors worn on a headband to pick up minute changes in electrical activity in the forehead muscles and within the brain. Currently such systems are only able to detect quite coarse changes in alpha and beta states of the user's brain, but future systems may conceivably allow mind control of computer systems, for applications such as thought-to-text word processing.

Ultimately these computers may move beyond simply sensing their environment and actually transform it in some fashion. Fletcher [Fletcher 1996 - #49] proposes a

universal force I/O panel that could form the shape of the Rosetta stone, a prehistoric fossil, a keyboard, etc. as necessary for the task and data at hand.

1.2.1.3 New Communications

Many of today's desktop computer applications are reliant on a connection to Intranet or Internet based resources, and the same is likely to be the case for the new breeds of computing device. Of course, for these new devices, which as a rule tend to be very mobile, some form or wireless communications rather than a fixed cable connection is required.

Early wireless communications technology was characterised by low bandwidth and low reliability. Although this is now much better and is continuing to improve, the concepts proposed to address these characteristics are still useful. For example, Dix [Dix - #34] argues that it is PACE that is generally the most important characteristic in wireless communications and not bandwidth, i.e. the importance of interaction speed and response times, not the speed at which large quantities of data can be conveyed. Dix also puts forward the notion of designing applications with the level of connectivity exposed to the user so that they may adjust their expectations appropriately.

New wireless communications technologies are coming from two distinct development efforts:

i. Those focused on improving wide area telecommunications networks that were initially intended to support mobile voice telephony, but which are now increasingly supporting data centric communications.

ii. Those focused on developing wireless successors or replacements of traditional local area networks such as Ethernet.

The majority of today's mobile telephone networks are second-generation systems; digital rather than analog (i.e. first generation) but still circuit switched (i.e. calls to a particular destination are routed and fixed for the duration of a call). GPRS (General Packet Radio Service) is extending this system to what is know as "2.5G", facilitating a packet-based data communication system where there need be no fixed calls in order to transmit data. As users only pay for the data they transmit, and not the duration of a

call, this enables an "always on" Internet connection. Combining this with the much higher bandwidths expected from third generation systems (3G) makes for a very comfortable communications link. See [Emmerson 1998 - #37] for a good overview of 1G to 4G telecommunications technology.

From the local area networks perspective, wired technologies such as Ethernet have made the jump to wireless implementations with standards such as 802.11 (see [Goldberg 1995 - #56] for a good review of 802.11). Wireless LAN systems differ from the telecommunications based systems in that they offer much higher bandwidths over a limited area (approximately 150 metres with most 802.11 implementations). Wireless LANs enable the ad-hoc formation of networks, but are more typically used to provide extensions to a core wired LAN (such as Ethernet) via wireless access points (essentially a bridge between wireless and wired networks).

A new form of wireless network has emerged with the specific intention of addressing the needs of small and mobile computing devices: Personal Area Networks (PANs). PANs intend to enable the various devices that a user carries on his person to seamlessly connect and communicate with each other. Bluetooth [Bluetooth-Consortium 2001 - #11] has become the industry standard for this form of short-range high-bandwidth networking (typically less than 10 metres range). Not only does Bluetooth define the means for communication, it also provides a discovery protocol that enables devices to query their physical environment and then discover and connect to nearby resources that they would like to use. For example, a user's mobile phone may connect to their PDA's address book in order to retrieve a phone number and then connect to the user's wireless earpiece to route the received audio channel. Applications need not be restricted to devices present on a single user. For example, consider an automatic check-in system at airports in which the user's PDA automatically discovers the check-in system and performs the necessary tasks for its owner.

Wireless LANs and PANs tend to support local area "hot spots" with high bandwidth and ad-hoc networking, whereas wireless telecommunications provides low bandwidth connections but with much broader coverage over a large part of the urban world. The technologies complement each other and will be used in conjunction. Mobile phones and PDAs seemed destined to merge into hybrid devices such as personal

communicators, and the Bluetooth consortium aim to make Bluetooth chips so cheap (around 5 dollars a piece) that there will be no obstacle to their goal of embedding them in the vast majority of electronic devices and appliances. Due to the nature of its design 802.11 technology it is more expensive to manufacture and also consumes more power, therefore it is likely to remain as an alternative to wired LANs for larger devices such as laptops.

One final interesting aspect of wireless communications technology is the additional ability to use it as a form of location sensor. A number of companies are working on solutions to provide location information to an accuracy of approximately 70 metres via triangulation of wireless telecommunication base [Duffet-Smith 1996 - #35]. This research has been expedited by the legal ruling in the USA that mobile phones must be able to report their location on emergency service calls (known locally as 911 calls) from 2002 onward [FCC 1996 - #39]. Similar triangulation techniques may be employed by wireless LANs and PANs, but the environmental effects on the signal propagation make it rather difficult in most environments. However, in PANs such as Bluetooth the cell size of the network may be so small, e.g. 10 metres, that the cell ID may be useful location information in itself.

1.2.2   Applications

These new technologies are interesting in themselves, but it is really the compelling new applications (or lack of them) that determine their success. One of the first application areas to be investigated was bringing the WWW to connected PDA devices. Early research in this application area concentrated on porting the same web browsing application semantics to these new platforms, with a focus on the effective caching and redistribution of client and server side processing to suit the new environment of use (i.e. poor resource devices with low bandwidth and unreliable connections) [Bartlett 1995 - #7] [Gessler and Kotulla 1995 - #55] [Kaashoek, Pinckney 1994 - #69]. Work by Voelker and Bershad [Voelker and Bershad 1994 - #144] actually sought to change the web browsing semantics in order to offer browsing experiences more relevant to mobile users. Their key inventions were the active document and dynamic URL supported in their modified web browser, Mobisaic. Special dynamic environment variables were made available to applications so that they could determine elements of

their context of use, such as location. References to these dynamic environment variables could be placed in a URL to make a dynamic URL. When submitting the URL request the web browser would substitute the references for the current environment variable value, enabling the context-aware selection of a web page. In addition to returning conventional web pages, the server may also return active documents. These appear to the user as normal web pages but they contain subscriptions to dynamic environment variables with associated actions to perform should the variable change, e.g. to reload the current URL or to load another URL. The combination of active documents and dynamic URLs allowed a context-aware browsing experience where pages appropriate to the current context could be downloaded and autonomously update their content with respect to further changes in context. Schilit et al also proposed a form of context-aware extension to web browsers [Schilit, Douglis 1996 - #126], but concentrated more on the context elements of cost and connectivity, making them very visible to users to enable them to make time/money trade-offs given their immediate information needs. Numerous scenarios for mobile web browsing have been posed, ranging from accessing the library whilst sitting by the campus lake (in Fosters's "Library without a Roof" [Foster 1995 - #51]) to in-car Internet access (in Jameel et al's "Web on Wheels" [Jameel, Stuempfle 1998 - #67]).

Continuing the transport theme, researchers at BT Labs near Ipswich, UK, have equipped busses on one of the town's main bus routes with GPS receivers that periodically transmit their location back to a central monitoring station [Thompson, Sheat 1999 - #142]. Their aim is to overcome what has been described as the highest barrier to the use of public transport: a lack of accurate, reliable and accessible information. Using the location information collected in realtime from buses they address this problem by providing ubiquitous access to a dynamic timetable. Through a web page, mobile phone, or intelligent bus stop display, a potential bus user can quickly find out which bus to take and when it will *really* be at their nearest bus stop. BT hopes this system will increase the use of public transport by locals and visitors alike.

1.2.2.1   *Electronics Guides*

Addressing the same type of potential user, electronic guides for tourists and/or residents are one of the most widely researched applications of new mobile technology.

They provide the means to research context-aware issues in an environment and application that everyone can immediately understand and in which new benefits can be clearly distinguished. Researchers at the University of Lancaster have been forerunners in electronic tourist guide research and have developed a system called GUIDE, based on a tablet computer that uses an 802.11 network for both wireless information access (via a web browser) and in determining location (from the cell ID) [Davies, Cheverst 1999 - #24] [Cheverst, Davies 2000 - #21] [Cheverst, Davies 2000 - #22]. The system aims to provide the user with maps for navigation, personal tours, online chat, online service provision and, of most interest to us, context-aware information retrieval. By clicking on a special information button on the GUIDE systems web browser interface a list of information related to the current context is displayed. In this system context is divided into environmental (time of day, opening times of attractions, etc.) and personal (visitor's current location, interests, language, food preferences, etc.). The context information is used in many places, not just in the information button function but also for tasks such as dynamically rearranging a tour itinerary when it becomes clear that a user will not arrive at an attraction before it is closed for the day. In addition to developing an object model to store contextual information, a major focus of the research has been on user interface design and evaluating the application through a real world implementation and user trial. Extending an existing application (i.e. the web browser) for a new purpose gave rise to a number of interesting issues, such as the semantics of the forward and back buttons (i.e. do they mean forward and backward in terms of pages or locations?). Issues of connectivity were also explored within the user interface, where it was found that exposing the connectivity level to the user greatly helped their understanding of the functioning of the system.

The CyberGuide system developed at Georgia Tech [Abowd 1997 - #1] [Long, Aust 1996 - #84] shares GUIDE's aim of reusing information that is already on the Web rather than creating new information specifically for an electronic guide, although they also see the need for active content with which the user can communicate or interact with the environment. Rather than a generic guide for a large area, as developed at Lancaster, the CyberGuide work has developed specialist guides for specific scenarios, e.g. an indoor Apple Newton-based guide providing information and navigation assistance to visitors at the university's open days (using infrared beacons to determine location), and an outdoor guide that helps the thirsty in finding the city's bars (using

GPS to determine location). A major focus is the development of a generic application structure comprised of modular map, information, positioning and communication components, which correspond to the real world human roles of cartographer, librarian, navigator, and messenger. The decoupling of these generic components allows for specific guide applications to be rapidly developed. In particular, the authors highlight the need to separate positioning and communication components (even if to begin with an application may use the same underlying technology to achieve both services) in order to enable the incorporation of different technologies in the future.

The TOI system (Travellers Online Information system) [Dias, Santos 1996 - #33] attempts to tackle a much wider-ranging form of electronic guide that provides countrywide travel assistance. They TOI authors describe their work as attempting to bring ubiquitous computing to GIS (Geographic Information Systems). TOI seeks to provide information about places, transport suggestions, digital maps with route guidance, general sport and cultural info, and services and goods acquisition (theatre tickets, cinemas, restaurants, car rental, cabs, etc.), where the provider knows the user's location automatically. The research concentrates on the development of such services and uses a relatively simple technical infrastructure where permanent information is stored locally on the user's device and volatile information (like event information) is delivered over an intermittent GSM connection. Location information is obtained via IR beacons, although it is intended that these may be substituted with other location systems. A novel user interface control is also presented, hierarchic marking menus, which effectively combine the two goals of ease of user interface navigation by new users with fast navigation by experienced users.

The work of Not et al [Not, Petrelli 1997 - #94] [Not and Zancanaro 1998 - #95] concentrates on the specific domain of museum guides, with the research focusing on the presentation and structuring of information. Rather than displaying information on a PDA screen, and therefore competing with the museum exhibits for the user's visual attention, the guide presents information audibly. The choice of this mode in itself presents a whole new set of HCI challenges, which the authors attempt to address. They examine how the "world" of information relating to exhibits may be structured, segmented and automatically presented to the user in an appropriate manner, suggesting an augmented world comprised of both physical and virtual elements.

Similarly, the Vuman series of wearable computers developed at Carnegie Mellon [Smailagic and Siewiorek 1996 - #132] present campus map and navigation assistance to users via a head-mounted see-through display at the same time they are viewing their physical environment.

1.2.2.2 Medical Applications

Another relatively large area of applications research for new types of mobile computing technology is in patient treatment. As with visitors in tourist guide systems it is clear that the doctors and nurses involved in patient treatment are highly mobile people and require access to information in a variety of places. One of the three application areas that Bruegge explores [Bruegge and Bennington 1996 - #18] is in the treatment of patients in their homes (the other two application areas are emergency service location-based information assistance, and transport inspection and maintenance). The simple ability to retrieve remote patient data (with no contextual assistance) improved the visiting doctor's productivity by twenty percent.

Within the confines of a hospital Freedman and Pollock [Freedman and Pollock 1996 - #52] implemented a PDA based wireless information system that gave doctor's access to patient and reference data during their ward rounds, which resulted in an increase in the quality of data collected. Similarly, Pentland et al [Pentland, Petrazzuoli 1997 - #105] seek to provide the Doctor with continuous access to reference materials, and also to offer the opportunity to communicate with other specialists. Where their approach differs is in the method of delivery: a wearable computer rather than PDA. This combination of computer and doctor they refer to as the "digital doctor".

Medical applications tend to concentrate on the provision of appropriate patient and/or reference information whenever and wherever it is needed via wireless communications and new mobile computing devices. Unlike tourist guides these applications tend to focus more on ubiquitous access to any information in any place rather than access to particular information in particular places. However, although not location oriented there are still many interesting contexts that context-aware software may use in this field, such as the nearest patient, the vital statistics of this patient, their treatment history, etc.

1.2.2.3   Augmented Reality

Some of the latter tourist guide application research we presented spoke of combining virtual data with the real world. This has become an area of application development in its own right, called augmented reality. Feiner et al [Feiner, MacIntyre 1993 - #41] helped pioneer augmented reality research with their work on merging X windows with reality via see-through head-mounted displays. The system monitored the location and direction of the user's head and enabled them to position windows so that they appeared attached to the display, attached to the direction of view (providing a virtual cylindrical workspace around the user's head), or attached to physical objects. The concept of merging the physical and virtual world was also implemented in a much more literal sense by Feiner et al in the KARMA project [Feiner, MacIntyre 1993 - #43]. This work explored how to merge virtual 3D models and diagrams with their real world counterparts in order to aid and instruct maintenance workers in their repair work. A main research theme in this literal form of augmented reality is in the precise tracking of the user's position and point of view, and in convincingly overlaying the computer generated imagery onto the user's view of the world. Webster et al [Webster, Feiner 1996 - #148] have also used the same technology to reveal more about the real world rather than to superimpose virtual data onto it. In a system designed to aid methods for construction, inspection, and renovation of architectural structures, they provide in effect a form of x-ray vision that enables inspectors to see features and structures that lie behind the surface layer of concrete, e.g. imagine being able to see the electrical wiring and piping behind a wall before beginning commencing reconstruction work there. Butz et al [Butz, Hollerer 1999 - #19] have sought to combine augmented reality with conventional computing devices such as laptops and PDAs in the context of collaborative working. The augmented reality view of a workspace allows users to work together on shared data, but it also allows them to maintain their own private view of the workspace for data they do not wish to reveal. Data may also be dragged and dropped to and from conventional computer displays so that the most convenient interface for manipulating this data may be utilised. Feiner et al [Feiner, MacIntyre 1997 - #42] also utilise handheld computers in their augmented reality tourist guide, overlaying indicators of interest onto the visitor's field of vision but using the PDA's screen to present the information content if an indicator is selected. This work is also

extended into a full multimedia "situated documentary" system in [Hollerer and Pavlik 1999 - #62].

Fitzmaurice et al [Fitzmaurice, Zhai 1993 - #48] concentrate on handheld displays that are tracked in order to provide a portable virtual reality workspace. These virtual 3D workspaces, described as fish tanks or 3D spreadsheets, are viewed and manipulated through a handheld display that when moved or tilted gives the effect of moving a window onto the virtual world. In further work on "Situated Information Spaces" Fitzmaurice et al [Fitzmaurice 1993 - #46] [Fitzmaurice and Buxton 1994 - #47] propose that these virtual workspaces can be situated in particular places within the user's physical environment whereby physical objects act as anchors for virtual information. One of the main aims of this research is to overcome information overload. Anchoring data to physical locations provides a logical means of partitioning and organising information, where the physical objects also serve as a retrieval cue to users. Although it is suggested that some physical indication of embedded information be provided, much like the flashing light of an answering machine, for the most part this research attempts to extend and complement existing physical infrastructure with virtual information. A simple example is given in which paper-based resources, such as a map, may be enhanced with more detailed information as the user's palmtop display is moved closer to the paper.

Rekimoto and Nagao [Rekimoto and Nagao 1995 - #112] effectively address many of the practical issues needed for widespread implementation of such systems with their work on identifying real world objects through the use of an enhanced form of paper barcode. These barcodes can be attached to physical places and/or objects and identified by some image recognition software running on the user's camera-equipped computing device. They propose application scenarios such as exhibitions of paintings where the information about a painting can be tailored to each user depending on their knowledge level, age, and preferred language. In [Rekimoto, Ayatsuka 1998 - #111] the authors propose the term augmentable reality to reinforce their research into augmented reality systems where the emphasis is on the user creating information rather than discovering and accessing existing data. In this work they concentrate on interfaces that allow users to tag data and voice annotations to location and time

contexts in an effective manner, not just through their portable devices but also through desktop email and web-based applications.

Such dynamic content is also the focus of Gellersen's research [Gellersen, Beigl 1999 - #54], in which the concept of environment-mediated communication (EMC) is developed. EMC enables person-to-person communication using the environment as a form of intermediary. The authors avoid particular implementation methods and concentrate on developing the semantics of how the physical environment could mediate communication. For example, a note attached to a door could be addressed to a particular person or a group of people, be relevant only during a particular time period, have one-time or many-time delivery semantics, be from an anonymous source, etc. They also expound upon potential storage and communication models from source to sink.

Brown [Brown, Bovey 1995 - #16] [Brown 1996 - #15] uses the Post-It note as a generic metaphor for augmented reality applications. Instead of writing notes on a piece of paper and attaching them to a physical object or surface, electronic Post-Its, called stick-e notes, can potentially contain any sort of digital data and be attached to a variety of contexts. A common structure for marking-up data with context information using SGML is proposed, and triggering (the automatic invocation of a stick-e note in a matching current context) is considered as the focus of application processing. The research concentrates on these generic data-structuring and context-matching issues of augmented reality systems, irrespective of devices, sensors or infrastructure used in particular implementations, and aims to satisfy a broad range of applications that require the ability to discover and present information in context. This research forms the foundation of our work on exploring context-aware application frameworks in chapter two, and also provides the basis for some of the fieldwork applications discussed in subsequent chapters.

As with the EMC work, stick-e notes are primarily a method of linking conventional digital data to the physical environment rather than offering new types of content or attempting to integrate the content with the physical world as in the more literal types of augmented reality systems. In a thorough overview of the role and applications of wearable augmented reality systems, Starner et al [Starner, Mann 1997 - #134]

accurately refer to this type of application as physically-based hypertext. The authors also describe another type of augmentation, that of extending a person's memory.

1.2.2.4   Memory Prostheses

With a belief in a future of "intimate computers" (i.e. wearables or PDAs that are worn or carried everywhere) Lamming and Flynn [Lamming and Flynn 1994 - #77] first explored the idea of using such devices as aids to human memory. They developed a system called Forget-Me-Not, based on the ParcTAB devices. This system observed the context of the user with respect to the documents that they worked with on their devices, automatically recording contextual biographies for each electronic object, which could be later used to help the user recall the information. Based on the observation that users rarely file or categorise short notes effectively, their contextual history, in terms of location, encounters with others, work station activities, file exchange, printing, and telephone calls, could all be used to help the user find a document, e.g. to help me find the note I made after receiving an SMS message from Petra.

This type of application is unusual from a context-awareness point of view as it is a historical context rather than a current context that is of primary importance. However, in other work on developing a "human memory prosthesis" the current context is of more concern [Lamming, Brown 1994 - #76]. In this work the authors separate memory problems into three categories: retrospective (remembering things about the past), prospective (remembering to remember things in the future), and action slips (forgetting to do things). The second two problems require the memory prosthesis to continually examine the user's current context and present information that should be remembered in the appropriate situation, e.g. the right time, place, encounter, etc.

Rhodes' [Rhodes 1997 - #116] work on a remembrance agent cites the key wearable computer characteristics of being portable while operational, hands free use, equipped with sensors, proactive, and always on, as a good match to the needs of memory prostheses. The remembrance agent continuously runs on the user's wearable and "watches over their shoulder" at what they are doing in order to present them with previously recorded notes, emails, papers, and other textual documents that may be of use to the user in their present activities. The agent uses seven contextual cues to gauge

the relevance of a document: the information itself, the wearer's physical location, the people around, a subject field, a date-stamp, a time-stamp, and the day of week. New types of context can be added, can be extracted from the documents themselves, or can be manually entered by the user as well as being automatically captured from sensors. An interesting and somewhat ironic observation is that the importance of different context elements changes depending on the current context. To cope with this observation Rhodes permits the weighting of context elements, and enables these weightings to differ for different types of activity and different types of document being recalled. The author also addresses how best to display the recalled information, balancing the amount of information available with the possibility of overwhelming the user. An interesting possibility posed by this research is the ability to use someone else's memory, accessing a different pool of information from another memory prosthesis [Rhodes and Starnar 1996 - #117].

Desktop-based memory prostheses such as Margin Notes (a program that tries to establish interesting links between the web page a person is reading and their various stored documents, notes, emails, etc.) [Rhodes 2000 - #115] show how context-awareness is not just useful in the domain of mobile devices. In these systems context is considered to be the current topics of the information being viewed, and the interests of the user, rather than the current physical location.

The CyberMinder system [Dey and Abowd 2000 - #32] concentrates on supporting the action slips or to-do items type of memory. Current to-do list programs help the user record to-dos so that they will not be forgotten, but they do not assist the user in remembering them in the situation in which they should be done. Some physical systems, such as Post-It notes, attempt to address this by placing the note in a particular location, but the authors argue that a much richer range of context is required to effectively support to-do reminders, e.g. location, time, share price conditions, weather, activity level, number of people present, etc. They also incorporate a degree of ubiquitous computing philosophy to the system with the ability to create and deliver to-dos from a number of different devices, and to allow third parties to also submit reminders to an individual.

1.2.2.5 Mediated Reality

Much of the work we have addressed thus far has dealt with accessing information in new ways, and in situating that information in the physical world. Another body of research lies in mediating reality, that is, in some way changing or adjusting it. Mann [Mann 1996 - #87] has worked on wearable computer devices over a number of years including much work in the area of augmented reality. He suggests the term "mediated reality" to emphasise that these devices can also alter and remove from the user's view of reality as well as adding to it [Mann 1997 - #88]. Rather than a see-through head-mounted display, one of Mann's devices uses opaque glasses that present the images from a head-mounted video camera (situated so as to record the view that the user would see without the glasses) but passes this through some sophisticated image processing software that may alter parts of the image first. For example, Mann poses the possibility of removing his ex-girlfriend from the scene whilst passing her by in the street!

The concept of synthetic synesthesia proposed by Mann is also explored by Foner [Foner 1997 - #50] in work on artificial synesthesia via sonficiation. The idea is to extend a person's senses by remapping those senses that are normally undetectable by humans onto senses that are detectable. Foner has remapped the different wavelengths of the invisible spectrum onto different wavelengths of the audio spectrum, enabling users to hear what they cannot see. Some interesting applications are proffered such as hearing if the grass is sick today, or "seeing" an army jeep in the jungle (because the jeep is camouflaged, and not really jungle, it only looks like the jungle in the visible range of the spectrum).

1.2.2.6 Ubiquitous Computing

Although some applications try to incorporate aspects of the ubiquitous computing philosophy, the majority discussed thus far are centred around wearable computers or PDA devices rather than ubiquitous computing's true vision of hundreds of devices transparently embedded in our everyday world. This is more for the practical and cost reasons of installing computing devices throughout the environment rather than a disbelief in the vision itself. However, there is work being carried out on real ubiquitous computing devices, primarily with the "Things That Think" research at MIT [MIT 2001

- #90]. Resnick [Resnick 1996 - #113] introduces this research in where he argues the case for providing computing intelligence in everyday objects with the view that if we are given new tools and media, not only can we accomplish new tasks, but we begin to view the world in new way. Borovoy et al [Borovoy, McDonald 1996 - #12] give an illustrative example with their work on examining how social dynamics could be facilitated or otherwise affected through an electronic badge that understands something about its wearer. When two people meet their badges communicate and a series of LED lights give each person an indicator as to how much they do and do not share in common.

Rather than equipping *people* with transparent computing capabilities, Paradiso [Paradiso 1996 - #97] explored how interaction and intelligence could be added to commonplace inanimate objects in our environment. As a simple but illustrative example, a number of party balloons were developed with the ability to interact conversationally. Upon detecting a sound a balloon would "arm" itself and then begin "talking" after the noise level had dropped below a certain threshold. A number of amusing demonstrations are described where the balloons interact not only with people but also with each other, such as two balloons playing each other excerpts from "duelling bango" and others reciting random quotes of Marvin Minski. Sound is the important context for this application, but for toys and tools Verplaetese [Verplaetese 1996 - #143] explores how location can be used. Rather than concentrating solely on externally referenced forms of location, where position is measured relative to the surrounding environment, Verplaetese examines how internally referenced location detection may be employed through sensors that detect acceleration, rotation, translation, etc. Resnick [Resnick, Martin 1996 - #114] also explores ubiquitous computing in toys but emphasises that these devices can be used to control the environment as well as sensing it and simply being used themselves. Through a practical implementation of programmable Lego-like bricks, the author aims to make children think of themselves as designers and inventors and perhaps change the way children think about and relate to computers and computational ideas. In summary, the focus of ubiquitous computing can be as much about making people rethink themselves and their place in their environment as it is about the new technology itself.

Elrod et al [Elrod, Hall 1993 - #36] explore how to make the buildings that we inhabit more intelligent, with the focus on cost reduction through efficient use of resources. In a testbed implementation a set of offices was equipped with temperature, light, occupancy, and active badge sensors, in addition to computer controlled ventilation, lighting, and heating outlets. Utilising the sensors a building management system may customize the user's physical environment to their particular environmental preferences, and cost savings can be made through user selectable strategies for switching off the lights, computer displays, and other appliances, and for turning down the air conditioning or heating when offices are unoccupied.

Lewis [Lewis 1998 - #83] proposes that ubiquitous computing will first infiltrate our homes through information appliances. Rather than the complex and difficult-to-use desktop computer, "information appliances" will offer easy-to-use access to computing resources to sophisticated but computer illiterate users. The author suggests that these information appliances will take the form of Web enabled phones, televisions, and other consumer appliances.

In addition to providing the physical computer resources scattered around a user's environment, another important aspect of ubiquitous computing is in the provision of "soft" resources (i.e. data and applications) that the user likes to work with at any of those computing devices. Bennett et al [Bennett, Richardson 1994 - #9] describe their teleporting concept, where a user's X-Windows application can follow them to the nearest computer terminal. The concept utilizes an active badge network and a database that stores the location of the terminals, so that when a user presses a button on her active badge a server can destroy the old client window, find the nearest terminal, build the new X client, and connect it to the appropriate server. This technique makes use of a proxy server that sits between application client and server, which retains the state of the application so that when a client is destroyed and new one connected, it appears to the user exactly as they left it. Wood et al [Wood, Richardson 1997 - #152] extended teleporting to form Virtual Network Computing (VNC) in which they enabled access of the user's "home" computing resources from any Java-enabled Web browser. The client side is very thin; it simply transmits mouse and keyboard input and then draws the resulting screen changes sent from the remote computer. In VNC the proxy server sits between the Web browser's Java applet client and the server's operating system,

therefore allowing the user to access any software or data on the home machine. The client-proxy protocols are generic so that any Java-enabled client may connect to any platform on which a VNC server is running.

Rather than have the same applications follow the user from place to place as in teleporting, Fickas et al [Fickas, Kortuem 1997 - #44] are interested in specialist data and applications being made available in different places. In their NetMan system [Kortuem, Segall 1998 - #73] the user's wearable computer discovers information and small applications within the environment, so that the wearable computer's software tools and data are customized to enable the user to work within their current environment. Imielinski et al [Imielinski and Badrinath 1994 - #65] expound on a similar concept which, based on increasingly ubiquitous wireless network coverage and increasingly small cell sizes, allows the whole environment surrounding the user to be treated as large in-place database.

1.2.2.7   *Specialist Applications*

There are many specialist applications that are being developed to take advantage of new mobile computing and context-aware technologies. Asthana [Asthana, Cravatts 1994 - #5] has developed a personal shopping assistant that aims to help direct (or perhaps, if the shop is involved, misdirect) the user to the goods they are interested in and to allow them to make inquiries about certain items. Brody and Gottsman [Brody and Gottsman 1999 - #14] have also produced an Internet enabled prototype device that enables shoppers to scan the barcodes of goods to find out more about them. The interesting spin on this application is that can also perform a search of Internet retailers to try and find the same goods for cheaper prices. A further interesting development in the same field is the Shopping Jacket [Randell and Muller 2000 - #108], which combines beaconing from shops with location-sensing of shoppers.

Dey et al [Dey, Salber 1999 - #30] provide conference attendees with a personal assistant that helps in a broad range of activities from receiving the slides of the current presentation to monitoring the interest of their colleagues who are situated in parallel tracks. In addition to location some less commonly used forms of context are also employed, such as interest levels of other people, the logical place in the current presentation, and the user's personal interest profile. Stilp [Stilp 1995 - #136], however,

concentrates solely on location information determined via mobile phone networks to support a number of tracking oriented services such as automatic locating of emergency calls, location sensitive billing, vehicle fleet management, and package delivery monitoring.

In chapter three we explore our own specialist application area: archaeology and ecology fieldwork, where the very nature of the work calls for very mobile, context-aware and easy-to-use technology. Usability and the correct choice of interface forms part of much of the application-based research that we have presented thus far. It is of particular importance in the case of our fieldwork applications, which must cause as little distraction from the task at hand as possible. We have devoted chapters four and five to the development of some user interface guidelines and a usability study respectively. Hindus et al [Hindus, Arons 1995 - #61] illustrate the importance of selecting the right user interface for new types of computing devices by posing the humorous scenario of a Doctor who, standing in front of a terrified patient, uses her PDA's voice interface to record how bad the patient's condition appears to be and that amputation is probably the only option.

In the ParcTAB research a number of applications were provided that let users access and control information, computation and their physical and electronic environments. An important realisation of the ParcTAB team was that user acceptance depended on many factors: size, appearance, convenience, peer pressure, application types, and critical mass of applications. They found that in general people had established work habits that are a barrier to learning a new system. For example, a calendar application on the TAB was not as successful as expected as people already have traditional solutions to this problem, and new solutions that are as good or only marginally better (such as access to the online calendar) are not easily adopted. Other applications that solved a real problem, however, were compelling to users. We believe our fieldwork applications provide good solutions for such real problems.

1.2.3   Software Infrastructure

To realize these new applications for new computing technologies there is a need for supporting software infrastructure to support their development and operation. Initial research in software infrastructure for PDAs, wearables and other forms of mobile

computer focused largely on the accessibility and consistency of remote resources. Honeyman and Huston [Honeyman and B 1995 - #63] proposed a number of caching and replication strategies for accessing a remote file system from a mobile device that suffers from some degree of disconnection. Saldana and Cohn [Saldana and Cohn 1994 - #121] reverse the problem so that the user's most important files are hoarded on the mobile device. In this case it is only when the user is near a desktop computer (where there is probably better communications coverage anyway) that the files need to be remotely accessed (i.e. from the desktop PC). Mukherjee [Mukherjee and Siewiorek 1994 - #91] examine the restrictions in the communication and computing resources of mobile computers from the perspective of applications rather than files. Similarly to the teleporting work earlier mentioned in the overview of applications, they recommend the use of thin clients with only commands sent between the two in order to minimize the need for communication and local computation.

Much of the earliest work focused on *location transparency*, i.e. seamlessly providing the same communications, computation, and resources that the user is offered on their desktop computing environments. Noble et al [Noble, Price 1995 - #93] suggest a more realistic proposal, called application-aware adaptation, and a concrete implementation of their ideas in the Odyssey API, an extension to the UNIX operating system. The aim of this work is still to minimise the need for user intervention due to poor communications or computing resources, but to reveal the available resources to the applications so that they may adapt appropriately. They cite data fidelity as the most suitable axis of adaptation where different data types offer different fidelity attributes that can be adjusted to suit the available resources. For example, a video player application may adjust the frame rate, image size, number of colors, etc. of the video stream to make the best use of the available computing and communication resources. In later work they reveal other applications that must still try and operate in an application transparent way (such as their Coda file system), but suggest that application-aware adaptation is appropriate for applications that can negotiate their resource requirements with the operating system [Satyanarayanan 1996 - #123].

This leads us neatly on to software infrastructure for *context-awareness*, where it is this very adaptation and reaction to the environment of use that is the centre of interest. Schilit et al [Schilit, Adams 1994 - #125] pioneered research in this field, producing a

framework on which many new applications were built, such as a location-centric yellow pages (where, using a technique called proximate selection, a list of entries is ordered by proximity) and a location-based file browser (which let users store files in particular physical locations). Many issues associated with context-awareness were raised, such as how to continuously present this information to the user without disturbing them, and predicting user's actions by their historical context. Schilit's thesis [Schilit 1995 - #127] describes a software framework primarily oriented at supporting location-aware services and applications but also dealing with the contexts of present people and resources. Schilit provided an initial classification of different types of context-aware computing activity, such as proximate selection, automatic contextual reconfiguration (where the software adapts itself to the environment), contextual commands (where context can parameterize commands such as "print" with the user's context, i.e. "print here"), and context triggered actions (essentially "if-then" rules based on contextual conditions).

Other researchers have since tried to better define what is meant by context itself. However, although researchers agree on a common intuitive understanding of context, there has been little common agreement on a formal definition or classification. For example, Bowskill et al [Bowskill, Morphett 1997 - #13] divide contexts into spectral, spatial, temporal, and emotional, whilst Chevurst et al [Cheverst, Davies 2000 - #21] suggest personal and environmental classes, Schmidt et al [Schmidt, Beigl 1998 - #128] make two broad categories of human factors and physical environment which are further subdivided, and Dey et al [Dey and Abowd 1997 - #31] propose physical, action-based, and emotional categories. Rather ironically it seems that one's perception of context is rather dependent upon one's context! The most successful definitions are deliberately all-encompassing such as "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object" [Dey and Abowd 2000 - #28]. It seems that the context in context-awareness can be just about any characteristic; real, virtual or imagined.

Harter et al [Harter, Hopper 1999 - #58] focus on location-oriented contexts such as presence, proximity, and direction. In their work examining the anatomy of a context-aware application, they build instances of every level of technology required from the physical ultrasound sensor system to applications like teleporting. Supporting the

application software they have developed a highly optimised and robust service that provides an object oriented model of the world via an underlying relational database and the means for fast execution of location-based queries and events. They suggest location relationships (e.g. next to, in front of, nearby, etc.) are more useful than absolute locations (e.g. latitude and longitude, room name, etc.) and offer a sophisticated concept of auras for modelling the directional aspects of location (e.g. a monitor facing my direction can be used from a relatively long way away but a monitor facing away from me is completely useless). This infrastructure has been built with the active BAT sensor system specifically in mind, but there are many challenges in being able to accommodate the plethora of other location sensor systems in the same supporting software infrastructure. Leonhardt et al [Leonhardt and Magee 1996 - #80] [Leonhardt, Magee 1996 - #81] address just this problem in their work on integrating location sensor systems. Their approach maps location data from different sensor systems onto system-specific cell spaces that are in turn mapped onto a common sensor-independent cell space, called the zone space, which may also be split up into manageable domains. The authors also make a good contribution to security and privacy concerns in [Leonhardt and Magee 1997 - #79], issues that have otherwise yet to be addressed in much depth in this new field. Demers [Demers 1994 - #27] suggests that even seemingly innocuous data can infringe on privacy if enough is collected. Herzberg et al [Herzberg, Krawczyk 1994 - #60] examine ways in which a user will be able to "travel incognito" whilst Pftitzmann et al [Pfitzmann, Pfitzmann 1997 - #106] look at security from a different angle, examining measures to protect user's privacy if their personal device (and data therein) is lost.

Leonhardt provides a fine platform for gathering, presenting, and managing location-aware applications but, as Schmidt et al state [Schmidt, Beigl 1998 - #128], there is more to context than location. In their work exploring other types of context they consider elements of the physical environment, centred around infrastructure and conditions, and also examine the more human-centric areas of context such as identity, social environment, and task. As a simple practical example of a non-location context they show an extended Apple Newton PDA that is able to automatically switch between portrait and landscape modes depending on its physical orientation.

Another more peculiar example of context is used in the AROMA infrastructure, developed by Pedersen and Sokoler [Pedersen and Sokolar 1997 - #104]. This system aims to support a peripheral awareness of colleagues at a remote location. Sensors such as microphones and video cameras are linked to capture objects which in turn are routed through abstractor objects that transform the data into abstract representations such as bustle factors. These abstract contexts are sent over the network and yet again transformed at the remote end by synthesizer objects, which send the data to some abstract physical means of representation such as a model merry-go-around whose speed is in part dependent on the bustle factor. In the work of Chavez et al [Chavez, Ide 1998 - #20] it is the user's task and their location that take primary importance in a supporting framework developed to enable devices to help users manage their complex work patterns in which they may actively switch between many different activities.

Abstraction is a key ingredient of Hull et al's [Hull, Neaves 1997 - #64] situated computing service which, while citing the need to exploit many different types of context, in its initial implementation is primarily location based. The main aim of the service is to abstract applications from sensors by interpreting the raw sensor data (possibly converting and combining data in the process) in order to provide a reliable, stable and abstract view of the status of the current situation.

Other work has concentrated on providing generic application-level functionality in software infrastructure. Kalakota et al [Kalakota, Stallaert 1996 - #70] suggest that intelligent agents that may move from computer to computer will help the user in a variety of tasks, and provide services that are similar to roles of human jobs such as a librarian, investment advisor, and online shopping assistant. Long et al [Long, Kooper 1996 - #85] also offer an infrastructure based on human roles. Their software supports guide type applications with four components based on librarian, navigator, cartographer and messenger roles. However, in later research by the same group at Georgia Tech the need for a more fundamental form of context-aware software infrastructure has been examined [Dey, Abowd 1999 - #29]. In the most broad-ranging research of context-aware infrastructure, in terms of the variety of contexts examined and supported, they present a system based on the concept of widgets. They hope their development of context widgets will share the same desirable characteristics as the widgets developed for graphical user interfaces, i.e. separation of concerns, reuse, easy

access to context data through polling and notification mechanisms and a common interface. Their component-based infrastructure consists of context widgets, which collect and present sensor data, interpreters, which transform data into more understandable forms, and context servers, which aggregate data into logically related groupings (e.g. all context data relating to a particular person). These components may be plugged together in different configurations and distributed over a network.

Our own work on context-aware infrastructure starts with an application-level framework designed to support programs that act on changes in context in some way, e.g. presenting a document to a user when she arrives at a particular location. In the final three chapters of the thesis we examine the related work on software infrastructure in much more detail than in this introductory overview, and propose our own solution for providing an application-*un*specific "context information service". This work proposes that each computing device will maintain a model of the world, or more accurately, the subset of the world whose context is of interest to its human user or client software. The world model contains objects, representing physical or virtual entities, and their attributes, which represent the context of those entities. A number of software components are presented, and these plug the model into the real world and allow the conversion, manipulation, and management of context information, and its access by client software or users.

1.3 Where this Thesis Makes a Contribution

Although new forms of computing and sensor hardware motivate this thesis to a large extent, we are only interested in using them and exploring the new possibilities that they create rather than in developing new hardware technology ourselves. Our area of interest is in exploring a vertical slice of context-aware software spanning context-aware infrastructure, application-specific infrastructure, applications, and user interfaces, as illustrated in the following figure.
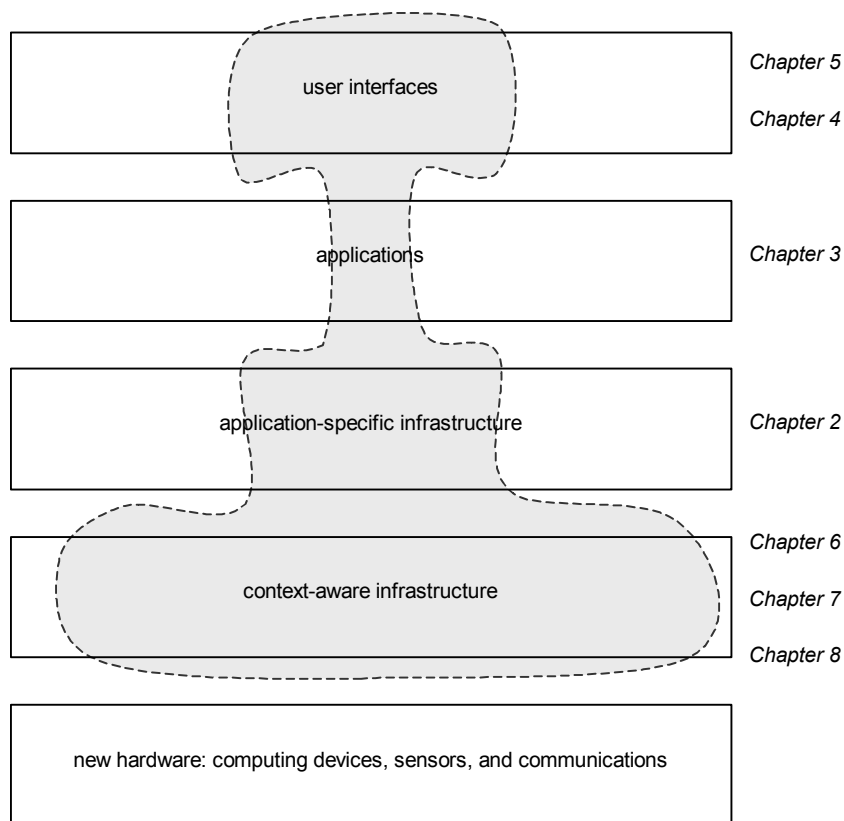
Figure 1. Illustration of the areas in which this thesis aims to contribute.

As can be seen from this diagram, rather than attempt to address all aspects of context-aware software we have focused our research in specific areas of each layer. The figure also shows which chapters address which layer. Chapter two starts at the application-specific infrastructure layer where we extend Brown's electronic Post-It note concept [Brown, Bovey 1995 - #16] [Brown 1996 - #15] in order to provide a framework to support applications that wish to react and adapt to changes in context. As suggested by the figure, this framework addresses a sizeable portion of context-aware applications but by no means does it span them all. Chapter three builds on this work by presenting a very specific application area, ecology and archaeology fieldwork, for which we have developed some context-aware PDA-based tools. This work in turn enabled our broader research into user interfaces, which we present in two parts: in chapter four we offer some HCI guidelines in building interfaces for such applications, and in chapter five we describe the results of a usability study conducted on the users of the fieldwork applications.

Having reached the top of our context-aware software stack we re-examined the role of the context-aware software infrastructure and, after analysing other research in this field, realised the need for a more general lower-level infrastructure to support context-aware applications. This work is presented in chapter six, followed by a description of a prototype implementation in chapter seven. Chapter eight concludes our work with an examination of how such a context information service might be distributed across a number of devices; unlike the rest of the thesis this is a paper design that has not been implemented.

We hope that our work on applications will demonstrate the exciting opportunities for new fields of computing – literally! We also use them as a basis to determine how useful the concept of context-awareness is in practical applications. Our usability study aims to show if such applications and technologies are readily accepted and useful to the everyday user for which they are intended. And from the good and bad HCI design lessons that we have learnt in developing the applications we also present some general guidelines to help designers and developers to build better user interfaces for the various new types of mobile computer.

Our work on infrastructure seeks to address the difficulty in adding context-awareness to applications, which we believe is the primary obstacle to its more widespread adoption. We propose two frameworks which we hope will enable developers to easily build rich context-awareness into their applications just as they may now easily build rich user interfaces into their applications via a common user interface framework.

Note that our research is presented in a chronological order. Although this means that a few ideas presented in early chapters are superseded in later ones, we feel that it will give the reader a better understanding of how the need for the different work has arisen and how and why the various layers of context-aware software have been distilled.

In summary, we hope that our work will demonstrate the new application possibilities which context-awareness presents and that by providing the necessary supporting architecture, user interface guidelines, and better understanding of context itself, will encourage, enhance and support the creation of context-aware software.

1.4   Published Work

Much of the work presented in this thesis has been published in conferences and journals. These publications are listed bellow against the chapter that they correspond to.

Published Work	Corresponding Chapter(s)
Pascoe, J. (1997). *The Stick-e Note Architecture: Extending the Interface Beyond the User*. International Conference on Intelligent User Interfaces 1997, Orlando, Florida, USA, 1997, ACM.	2
Pascoe, J., D. R. Morse, et al. (1998). "Developing Personal Technology for the Field." *Personal Technologies* 2(1): 28-36.	3
Pascoe, J., N. Ryan, et al. (1998). "Context Aware: the Dawn of Sentient Computing?" *GPS World* 9(9): 22-30.	3
Pascoe, J., N. S. Ryan, et al. (1998). *Human Computer Giraffe Interaction: HCI in the Field*. Workshop on Human Computer Interaction with Mobile Devices, University of Glasgow, UK, 1998.	4
Pascoe, J. (1998). *Adding Generic Contextual Capabilities to Wearable Computers*. The Second International Symposium on Wearable Computers, Pittsburgh, PA, USA, 1998, IEEE.	3, 6, 7
Pascoe, J., N. Ryan, et al. (1999). *Issues in Developing Context-Aware Computing Applications*. HUC '99, Karlsruhe, Germany, 1999.	6, 7
Pascoe, J., N. Ryan, et al. (2000). "Using While Moving: HCI Issues in Fieldwork Environments." *ACM Transactions on Computer-Human Interaction* 7(3): 417-437.	4, 5

Table 1. Published Work.

Chapter 2:

Stick-e Notes: a Metaphor for

Context-Aware Applications

Our work on developing context-aware software was initially inspired by Peter Brown's concept of *stick-e notes*, originally conceived as an electronic equivalent of a Post-It note [Brown 1996 - #15]. Instead of writing on a yellow piece of paper that one can subsequently attach to a desk, door, or some other object, the electronic stick-e note is written on a handheld computer (aware of its current whereabouts) that automatically 'attaches' the note to the user's current location. The flip side of this authoring of notes is their retrieval at a later date. With a paper Post-It note one cannot help but see the note when approaching the object that it is attached to (providing one's attention is focused on or nearby the object in question). With stick-e notes the user's handheld computer will automatically pop-up the note on the display or provide some other cue of the note's presence to the user as they approach the location in which it was recorded. This process of automatically retrieving pre-recorded notes associated with the current location is called *triggering*.

Unlike paper Post-It notes, the stick-e note metaphor is easily generalised. One can consider creating all sorts of digital information besides textual notes (e.g. audio notes, pictures, even computer programs, etc.) that could be attached to all sorts of context elements besides location (e.g. attached to a person, time of day, temperature range, etc.). Essentially, the stick-e note metaphor can be thought of as a general model that facilitates (i) the association of digital data with a physical context and (ii) its automatic retrieval through a knowledge of the user's current context.

This metaphor seems to fit many types of context-aware application. For example, an electronic tourist guide that presents the tourist with information on various attractions as they are approached, or a reminder application that prompts the user about a

question they wish to ask when they next meet a particular person, or an office environmental control system that invokes heating, air conditioning and lighting systems in response to changing levels of temperature and lighting, etc.

A core theme of our research is to explore how the development of context-awareness, which is currently prohibitively complex and time-consuming for many applications, can be supported through underlying infrastructure. The diverse applicability of the stick-e note metaphor made it an ideal candidate for such an infrastructure and the resulting stick-e note application framework was our first endeavour in providing generic context-aware support.

Another core theme of our research is to explore how context-awareness can be usefully applied. To this end we developed an 'electronic Post-It note' program based on the stick-e note application framework, serving to both evaluate the framework's ability to provide useful services and also to explore a novel application of context-awareness.

This chapter describes the issues in context-awareness that we have investigated through the development of both the stick-e note application framework and the electronic Post-it note program. Firstly, an overview of the design and operation of the framework and application is given. This is followed by a discussion of the issues that they have evoked, distilled into three categories: obtaining and representing context information, authoring content-dependent information, and triggering information in context.

2.1   Overview of the Stick-e Note Application Framework

The general capabilities required of the stick-e note application framework centre on two key activities: authoring and triggering of stick-e notes. From the perspective of developing the framework we are not interested in application-specific functionality, content authoring tools, user interfaces, etc. Rather, we are interested in providing the core stick-e note model and services to support applications, which will provide their own esoteric behaviour based on the general stick-e note metaphor.

The authoring and triggering of stick-e notes is a composition of three main processes:

➢ **Attaching:** connecting a piece of digital data to a context.

➢ **Matching:** checking if any stored stick-e notes are associated with the current context.

➢ **Dispatching:** sending matching notes to any interested parties.

Most of the research issues lie in the attaching and matching of stick-e notes, where we need to explore how to represent and process context information. The dispatching process deals with more conventional client maintenance tasks.

For all of these processes the stick-e note is the primary unit of information. It contains two general types of data:

➢ **Content:** the digital data of interest to the client.

➢ **Context:** the situation to which it is attached.

The content of a stick-e note is not of interest to us in designing the framework, we leave the client applications deal with matters of content as they see fit. The framework simply supports its storage in a stick-e note via a URL that the client can use to specify a particular digital resource. The context of a stick-e note, however, *is* of interest to us. It is the currency with which the framework operates. The framework should be responsible for the *capture, structuring, comparison, conversion and manipulation* of context elements in as transparent a way as possible to the client applications which, for the majority of the time, will only be interested in dealing with the stick-e note content rather than the context to which it is attached. There is a clear division of responsibility between client applications, which deal with all things content-related, and the framework, which deals with all things context-related.

2.1.1   Context Information

Let us consider more precisely what context functions will be needed in the framework to support the authoring and triggering of stick-e notes. Capturing context will involve communicating with external sensor devices (be they hardware or software) in order to establish the current values of different context elements. For example, the framework may communicate with a GPS (Global Positioning System [Texas-University 1999 -

#141] [Herring 1996 - #59]) receiver attached to a handheld computer to obtain the current location of the user. Of course, a user may supply the value of a context element manually (e.g. "I am at the library") but generally it is automatically captured context data that is of most value. Either way, the captured context element values need to be modelled in some form of structure within the framework, such as a latitude and longitude data structure to hold the location data generated from a GPS receiver. There may also be a need for higher level structuring of this data such as representing points, rectangles and polygons in the case of a location context element, and different formats in which it may occur, e.g. latitude and longitude, ordnance survey grid reference (within the U.K.), post codes, etc. To facilitate the matching of stick-e notes with the current context, mechanisms should be provided to compare, convert and manipulate these structures. Considering location data, we may want to check if two points match, to check whether a point is within a rectangle, to convert from latitude and longitude to ordnance survey grid reference units, to calculate the distance between two points, to move a point west by 100 metres, etc. Although we have only used the location context element as an example, the same general facilities described are required for all types of context element to enable the attaching and matching of stick-e notes with the current context.

There are potentially hundreds of useful context elements ranging from physical conditions, such as a temperature reading, to more conceptual conditions, such as the mood of the user. It is clearly beyond the scope of this research to develop support for all conceivable context elements, instead we concentrate on a few of the most useful ones: *location*, *with*, and *time*. This allows stick-e notes to be attached to *places*, *people (or other objects such as equipment)*, and *times of day* respectively. Although we concentrate on these context elements it is our aim is to develop a generic framework that will support the addition of many other types of context element at a later date. That is, *we are more interested in designing the general representation scheme in which different context elements reside rather than developing the individual esoteric context elements themselves.*
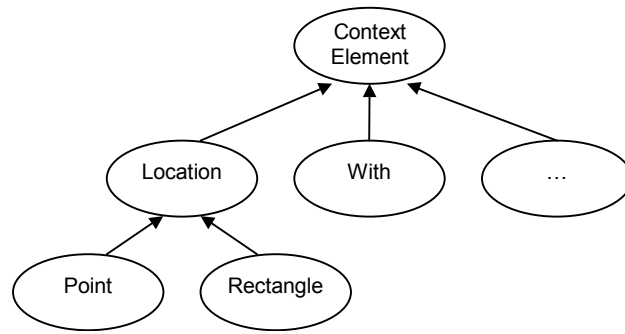
Figure 2. Simplified context class hierarchy.

We have considered context from an object oriented perspective and have formulated a class hierarchy headed by a generic context element class from which various types of more specific context element are derived, as is shown in simplified form in Figure 2. The top-level `ContextElement` class is essentially just a grouping class under which all the different types of context element reside. The second-level classes represent a particular type of context element, such as a location, and define the common interface that all the more specialised derived classes have to implement. Derived from these classes there may be several layers of specialisation of context element before reaching an instanciable class (the hierarchy maybe arbitrarily deep), but for the sake of simplicity we have derived the instanciable classes of `Point` and `Rectangle` directly from the `Location` class.

We can view the hierarchy as having four types of classes:

➤ **Base Class:** acts as an 'umbrella' class from which all context elements are derived.

➤ **Type Classes:** define the interface for particular types of context element.

➤ **Specialisation Classes:** define more esoteric versions of a general type.

➤ **Instanciable Classes:** the classes that can be instantiated and used to model the value of a context element. Note that in many cases there will probably be some blurring of the boundaries between specialisation and instanciable classes.

This hierarchy of different classes facilitates the incremental development of context support, where new types of context element and new implementations of existing types may be added by the application developer as and when needed. For the moment our hierarchy simply consists of location, with, and time related classes. The

hierarchical structure also makes it possible to deal with context elements at various levels of abstraction. For example, an underlying location polygon could be presented and manipulated directly as a location polygon class, or in the more abstract form of a general location class, or even in the completely generic form of the context element base class. This allows the framework and clients to work at high levels of abstraction and hence be able to seamlessly accommodate future additions to the context hierarchy that are added beneath their operating level of abstraction. Extensibility is desirable in many applications but absolutely critical in a context-aware framework because it is simply an impossible task to even conceive of all the different types of context element that may be useful let alone to provide support for them all.

In our framework we have separated the facilities for structuring, working with, and generally representing context information from those facilities responsible for the automated capture of context through communication with external hardware or software sensor devices. The aim of this separation is to disconnect context data from specific sensors, allowing for a more general-purpose context representation that can more readily support the addition of new sensor devices.

Similarly to the context elements, there is a plethora of sensor devices (in the form of both software and hardware) that can supply data about the current context. These sensors can be grouped into categories in much the same way as the context element hierarchy, i.e. by the type of context element data that they supply. This results in a sensor class hierarchy that somewhat mirrors the context element hierarchy.

Note that sensor classes should not be mistaken for the physical sensor devices themselves. They are merely pieces of software that provide a standard communication medium for obtaining the context element data provided by a particular type of sensor device. In fact, it is quite possible that we could have more than one sensor class representing the same physical sensor device if it can provide multiple types of data, e.g. a GPS receiver that supplies location, orientation and time context elements. In such a case the physical sensor device will be represented logically within the sensor hierarchy as three separate sensor classes, one for extracting location information, one for extracting orientation information, and one for extracting temporal information.
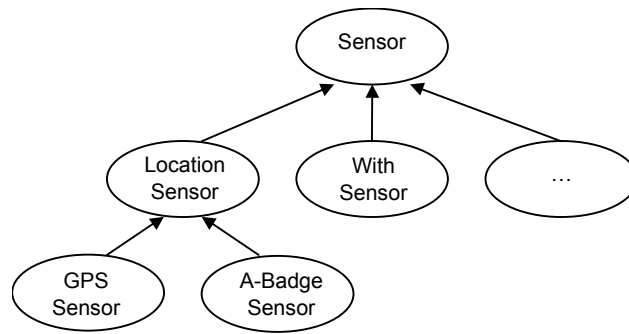
Figure 3. Simplified sensor device class hierarchy.

The construction of the sensor hierarchy is illustrated in Figure 3. As with the context hierarchy, there are four different types of class:

➢ **Base Class:** acts as an 'umbrella' class from which all sensor devices are derived and records the general state of the sensor device (if it has been checked, if it is not working, etc.).

➢ **Type Classes:** define the sensor interface for sensor devices that provide a particular type of context element. For example, a location sensor class could define a `GetLocation()` method that all derived location sensor classes would subsequently have to implement.

➢ **Specialisation Classes:** define more esoteric versions of a general type of sensor, i.e. they may provide additional interface methods specific to that class of sensor device.

➢ **Instanciable Classes:** the classes that can be instantiated and used to extract data from a particular sensor device. As with the context hierarchy, in many cases there may be a blurring of the boundaries between specialisation and instanciable classes.

Organising sensor devices by the type of context element they supply makes for a flexible hierarchy that can readily accommodate new sensors beneath a standard interface. The standard interface for each context element sensor, specified in the type and specialisation classes, allows sensors to be accessed in a generic fashion so that no extra programming effort is required to incorporate a new sensor device into a client program. For example, the location sensor class may define a `GetLocation()` method that can be invoked on any of the instanciable location sensor classes. The method will

return a location context element that may be a reference to any class of location objects. That is, the returned location may be in any of the possible forms, such as a `Point` or `Rectangle`, whichever is most appropriate for that particular sensor device.

The clients of the framework will often not be interested in communicating with specific sensors, they will simply want to retrieve the current value of a particular context element. For example, a client may enquire as to what their current location is; from the client's perspective the type of sensor device used to retrieve that location is immaterial. That is, clients often want to work at the level of acquiring the current value of a context element rather than at the level of finding, instantiating, and communicating with sensor devices.

In order to achieve this level of generality we created a `SeEnvironment` object that provides access to the current value of context elements such as location, and which utilises all the available sensor devices to provide the best data transparently to the client. It is a centralised framework resource that any client may access through generic context element accessor methods, e.g. `GetLocation()`, `GetWith()`, `GetTime()`, etc. The client is also able to specify the specific type of context element that the returned value should be compatible with if need be.

 In addition to accessing context elements automatically captured from sensors, the `SeEnvironment` object also manages *pretend contexts* that allow the user to simulate the value of a context element, e.g. to pretend to be in a particular location. This is useful not only for supplying values of context elements that cannot be easily automatically obtained (e.g. capturing the user's emotional state) but also in enabling the user to preview or review stick-e notes that will trigger in a specified context by pretending to be there now. There are differences in representing pretend contexts as opposed to real ones due to the fact that they need not necessarily obey the same laws of physics. In the case of location context elements there is no reason why a user could not pretend to be in two or more discrete places at once, indeed, this ability can be most useful. For example, pretending to stand outside of the fish and chip shop in Trewoon and the Cornish pasty shop in Truro in order to peruse and compare the information presented by an electronic culinary guide of Cornwall.

2.1.2   The Stick-e Note

We have described how the framework represents context information, how it captures it, and how it makes it accessible. Now we describe how context and content are brought together to form a stick-e note via the `StickeNote` class. This class consists of only a single text field with which to specify the content because, as was mentioned earlier, content can be represented as a URL, which the client application is responsible for interpreting. The context part of the class is a more complicated matter. It is defined by what we call a *trigger condition*, i.e. the context under which the stick-e note should be invoked. The trigger condition consists of a number of *trigger contexts* that can contain any number of context elements of the same general type. A stick-e note's trigger condition may consist of many such trigger contexts, allowing stick-e notes to be attached to more than one type of context element. For example, in creating a 'go home' message that should be triggered if the location is the computing laboratory, the date is December 24th, and it is after 11pm.

The automatic delivery of stick-e notes through triggering is the main interest of our research. However, other work by Brown [Brown and Jones 2001 - #17] investigates other methods of retrieving information via context.

A stick-e note is triggered when at least one context element in all of the trigger contexts is met, i.e. the context elements within a trigger context are logically ORed together, and the trigger contexts within a trigger condition are logically ANDed together. Figure 4 illustrates these logical relationships between trigger contexts and their context elements. The horizontal groupings in the figure represent the separate trigger contexts that comprise the trigger condition, and its individual context elements. Using the example supplied in the figure, the stick-e note will trigger if any of the *location* conditions are met at the same time as any of the *with* conditions.

If a particular trigger context contains no context elements, such as *Temperature* in the figure, then it will not be used in determining the success of a triggering condition as the stick-e note is considered as having *not* been attached to any temperature. Should a stick-e note have no context elements or trigger contexts at all within its trigger condition then it will simply not trigger because it has not been attached to anything

(perhaps the stick-e note's content has been defined but it has not been attached to a context yet).
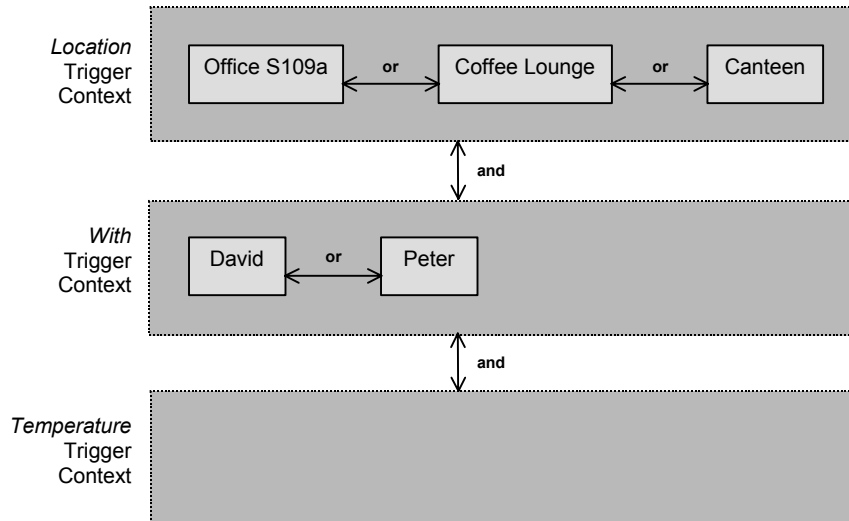
Figure 4. Example of the logical relationships within a trigger condition.

It is likely that sets of stick-e notes will be developed that relate to a similar subject, e.g. a set of stick-e notes developed on the archaeology of Zanzibar. A facility for representing these groupings of stick-e notes, known as *families*, is provided in the framework, thus allowing the user to organise a collection of stick-e notes into a more meaningful form than one huge pool of disparate notes. However, the concept of families is not convenient in all cases, especially for transient stick-e notes that are not stored for long, e.g. electronic post-it notes. Support for these transient stick-e notes is provided by a *universal family* into which all stick-e notes without pre-assigned families can be placed. Stick-e notes may also belong to more than one family – the families can be thought of as indices onto a global set of stick-e notes.

In effect families (implemented by a class called `SeStore`) act as a form of index onto a group of logically related stick-e notes. The references to the stick-e notes that a family contains can also be annotated with a *trigger-check-ready* status attribute which can be set to *yes* if the stick-e note should be considered by the triggering process and *no* if it should be ignored by the triggering process. The latter case is useful, for example, when developing a set of stick-e notes; during which time the developer may wish to deactivate stick-e notes that are currently in production. The assignation of the status attributes and sequence to references within the family rather than within individual

stick-e notes is a logical choice if re-use of stick-e notes within different families is considered. That is, a stick-e note may be used in more than one family and may appear in different sequences and have a different status in each one; modelling the status or sequence as part of the stick-e note would prevent such re-use.

To ascertain whether a particular stick-e note should trigger in the current context, a client can invoke the `Trigger()` method of the `StickeNote` object. The stick-e note will then compare its various trigger contexts with the current context elements retrieved from the `SeEnvironment` object. If they match within given parameters then the stick-e note will indicate that it should trigger. Note that some other party must invoke the trigger checking of a stick-e note: it is not an autonomous process.

2.1.3 Stick-e Note Processing

The stick-e note framework subscribes to the model-view-controller design pattern [Gamma, Helm 1995 - #53]. This design pattern separates the design and implementation of the user interface, i.e. the view, from the internal representation of what is being displayed, i.e. the model. A third entity in the pattern, the controller, acts an intermediary that 'drives' the model, i.e. initiating and controlling any processing activities. For example, a clock program could be separated into the display that draws the hands on the screen (the view), the internal representation of the clock's operations and current state (the model), and finally, the controller that starts the clock and keeps it ticking.

In the case of our stick-e note framework, the families of stick-e notes can be considered the model but there is no tangible view as such. Instead, an interface for a view is provided in a class called `SeDisplay`, which defines the interface mechanism for delivering triggered stick-e notes to clients. It is by inheriting the `SeDisplay` class that the clients of the framework may receive triggered stick-e notes. The manager, or `SeManage` object, which we now introduce, can be considered the controller. This manager is a centralised resource whose main duties are to supervise requests for the opening and closure of families and to regulate the invocation of the trigger checking process on the stick-e notes in each of the open families. Although individual `StickeNotes` provide a method to determine if the stick-e note can trigger it is

`SeManage` that must schedule and invoke it on all of the stick-e notes to be considered for triggering. `SeManage` also notifies the views of any relevant triggerings.
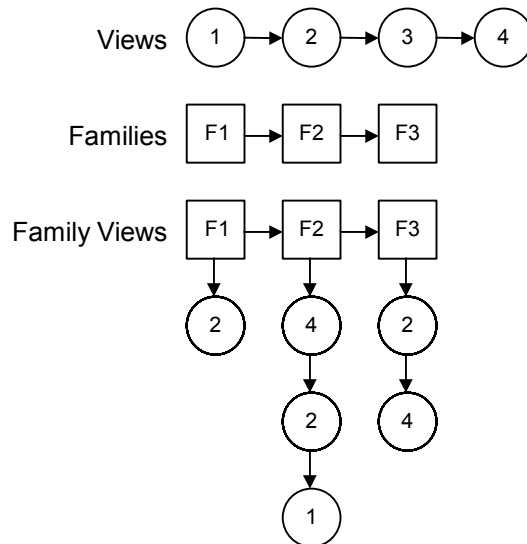
Figure 5. SeManage interest lists.

In order to manage the interconnections of interests between different registered views and open families of stick-e notes SeManage maintains three lists of information as shown in Figure 5. The list of views maintains information about the set of registered clients and the list of families maintains information about the sets of open stick-e notes. The final family-view list maintains information about what clients are interested in which families.

2.1.4 Framework Summary

Context elements and sensor devices are modelled in two independent but similar hierarchies that are organised into branches according to context element, enabling generic interfaces to be constructed for particular categories of sensor or context element. A `SeEnvironment` object presents some simple accessor methods to retrieve the current value of a general context element without the client needing to know the specific representation or sensors used to obtain it. The `SeEnvironment` object also has the ability to model pretend contexts, which may be used to simulate the current values of any context element.

The attachment of some digital information to a particular context is represented by the `StickeNote` class that specifies the content in the form of a URL and that defines the context in the form of a trigger condition. This condition is comprised of a number of trigger contexts, one for each general type of context element (e.g. location, time, temperature, with, etc.). If a trigger context is not empty then at least one of the values that it contains must match the current context in order for the trigger condition to evaluate to true, and hence to cause the stick-e note to trigger. Stick-e notes are individual objects but may be logically grouped into one or more families for management and organisation purposes. However, the stick-e notes remain separate entities in themselves in order to promote their reuse (e.g. a stick-e note describing the Cathedral could be included in several different Canterbury tour route families) and facilitate the dynamic creation and distribution of sets of notes (e.g. downloading a set of stick-e notes from a server which dynamically constructs a family group based on the contextual area of interest specified by the client).
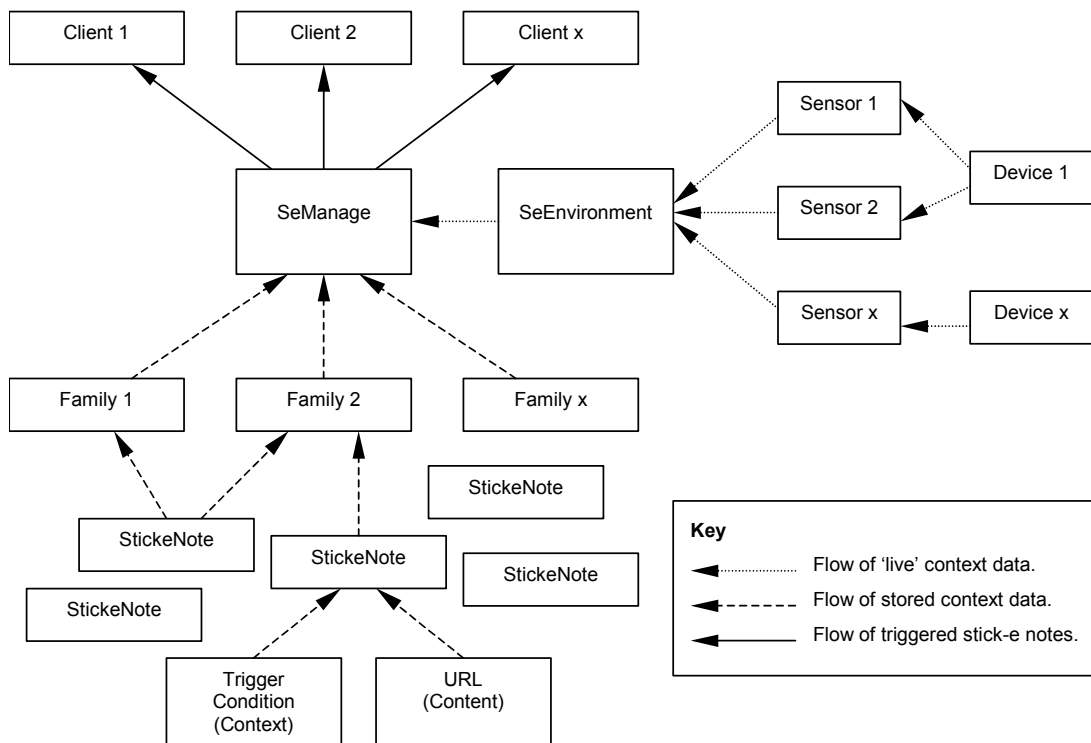
Figure 6. Framework components and the flow of context data.

The framework is based on a model-view-controller design pattern that de-couples the representation, display, and operation of a system. `StickeNote` and `SeStore` classes form the model, a `SeDisplay` class provides an interface for the delivery of triggered

stick-e notes to external views/clients, and a `SeManage` object takes the role of controller. The `SeManage` object manages the set of open families and the clients that are interested in them. It is also responsible for supervising the trigger-checking process, though it is the individual `StickeNotes` that perform the evaluation of their own trigger condition.

Figure 6 illustrates how the framework components are connected in terms of the flow of context data. 'Live' context data flows from physical devices into one or more sensor classes whose output is consolidated in the `SeEnvironment` before being dispatched, upon request, to the `SeManage`. The `SeManage` compares this live data with the context elements stored in the trigger condition of stick-e notes contained in any open families. Copies of any stick-e notes that trigger as a result of the comparison of live and stored context data are dispatched to any interested clients. More detailed objected oriented design diagrams for the stick-e note application framework can be found in appendix A.

To test out the viability of this framework, and also to explore the application of context-awareness in real applications, we developed an electronic post-it note program based on this framework. In the remainder of this chapter we present this application and the context-aware issues that arose as a result of this practical experiment in context-awareness.

2.2   Overview of the Electronic Post-It Note Application

The 'Electronic Post-It Note' program was developed as an experimental context-aware application that utilised the stick-e note framework. Our hardware platform for experimentation was a Hewlett Packard 200LX handheld computer linked to a Silva Compass GPS receiver (later replaced by a Garmin 45 GPS receiver). The handheld computer is essentially a miniaturised version of a desktop MS-DOS based computer, with very limited processor, storage, and screen-space resources.

The electronic Post-It note program we developed (similar in principal to the situated information spaces concept [Fitzmaurice 1993 - #46]) for the platform allows the user to write brief textual messages that are attached to their current context. At a later date these messages may be automatically re-displayed when re-entering that same context.

Essentially the aim of the application was to provide an electronic equivalent to the conventional paper-based Post-It note. This work allows us to not only to evaluate the stick-e note framework but also to explore some of the issues involved in developing a specific context-aware application.

Creating an electronic Post-It note is a simple matter: the user simply types in the textual message on the handheld computer and the program automatically retrieves the current context to which the note duly is attached. The electronic Post-It note is stored via a framework `StickeNote` object with the text message forming the content.

Figure 7. HP handheld computer linked to Garmin-45 GPS receiver.

To view Post-It notes that are attached to the current context takes no effort at all by the user. On start-up the program automatically configures the framework `SeManage` object to perform trigger-checking at appropriate intervals and proceeds to register itself as an interested party in receiving triggered notes. When the triggering process produces a triggered note `SeManage` sends a copy to the electronic Post-It program which reacts by sounding a brief beep to indicate to the user that a triggering has occurred. The note name is also momentarily displayed on the screen providing a notification to the user without disrupting their current work too much (equivalent to

someone sticking a Post-it note on your desk). This is what can be seen on the screen in Figure 7. It is then up to the user to decide their course of action in response, but presumably they will eventually wish to see the triggered notes, if only to be able to discard them. To this end they can either view a list of triggered notes or access a short cut in order to quickly display the last stick-e note to trigger. Similarities can be drawn with the design of email delivery systems. However, stick-e note delivery tends to have more urgency as it is of immediate and specific relevance to the user's current circumstances whereas arriving emails can normally be safely ignored until a later date.

It is tempting to equip the software with all sorts of features that would add useful function to the application. However, in doing so we would end up with a more complex interface that would not be so immediately accessible. Considering its paper-based counterpart, it is the simplicity and immediate usability of the paper Post-It notes that make them so attractive. It is just a simple matter of jotting down a note, tearing it off the pad, and sticking it on the desk.

We have tried to retain this simplicity within the electronic Post-It note program by providing a minimalist, yet functional, solution. There are only four menus, the longest of which has only three items, and most options (such as creating and attaching a new stick-e note) can be selected directly from the function key strip. The menus correspond to facilities provided for attaching and viewing notes (as described earlier), for controlling the trigger checking parameters (on/off, periods between trigger checking cycles, etc.), for monitoring the environment (adding pretend contexts and viewing the status of attached sensor devices), and for organising notes into families. The program attempts to provide these facilities through a simple interface that doesn't overwhelm the user, offering features in an intuitive and easily accessible manner.

The development of this electronic Post-It note application and the stick-e note application framework has posed many interesting issues in the design of context-aware systems. The remaining sections of the chapter discuss those issues, ranging from capturing context via hardware sensor devices, which is where we start, to aspects of user-interface design for context-aware systems, which is where we conclude.

2.3   Issues in Obtaining Context

A basic need of any context-aware program is to obtain and represent values for a context element. Our framework provides the building blocks to achieve this in two hierarchies, one for context elements and one for sensor devices. Although values for the majority of context elements are likely to be derived from sensor devices, in our framework we have been careful to keep representation and capture of context separate. This avoids polluting the context representation model with functionality that ties context elements to particular sensor devices, which would limit the flexibility and extensibility of both context representation and capturing infrastructure. Many of the sensor devices are also likely to be platform dependent, so it is best to isolate such dependencies as much as possible in the capturing infrastructure. Additionally, some clients do not require the use of sensor devices at all (e.g. on a desktop PC where pretend locations are being used to simulate a tour of Zanzibar) in which case we only require facilities to represent context and not to capture it.

Using these context element and sensor hierarchies a central framework resource, called `SeEnvironment`, provides an abstract interface between clients (which seek the current value of a type of context element) and the capturing infrastructure (which can provide the current value). It is easy to tell which context element the client seeks as they make it explicit in a request to the `SeEnvironment`. However, the `SeEnvironment` faces a harder task in working out which sensor classes it can use and/or communicate with. Currently it tries to instantiate every type of sensor class that it can find (using a default configuration) to supply the type of context element it is seeking. This is clearly not a very efficient model as there may be many sensor classes and perhaps, for example, only one which can be currently used with the single sensor connected to the computer. Additionally if the default configuration is incorrect (e.g. the GPS receiver was plugged into port 2 instead of port 1) then the device will be overlooked.

The problem is essentially that the `SeEnvironment` is not able to determine what sensors are physically present or available. This is a fundamental problem faced by any context-aware system that tries to be flexible in its approach to sensor support, i.e. how to find out what sensors are available and how to talk to them. We propose to solve this problem by making the user partially responsible for telling the framework what sensors it can use, as it is only the user who possesses such knowledge. This would be a

once-only operation performed when a sensor is first connected to the system, and would involve the user specifying the various configuration parameters of the sensor device. We also propose to make the device infrastructure more autonomous with devices modelled as independent programs that monitor a physical sensor device and inform the `SeEnvironment` of the context elements that they can provide. In this way the `SeEnvironment` can simply subscribe to the services offered to it by these *autonomous sensor monitors*. (This concept is explored more in the work described in chapter 4.) The role of the `SeEnvironment` is narrowed to that of providing a centralised access mechanism to the best current value for each type of context element. The need for the device hierarchy is thus reduced and may in fact be made redundant altogether. In the new model perhaps the autonomous sensor monitors will be written completely independently of each other with only an interface for publishing, subscribing, and elementary control in common.

2.4   Issues in Representing Context Information

Having captured the context information there are a number of factors to consider in representing it. With many sources of context there is a degree of error involved, such as the potential +/-100 metre inaccuracy in uncorrected GPS location data (prior to the removal of "selective availability" in 2000; the error is now much improved at +/- 10 metres, but mobile phone system positioning technology is still only accurate to approximately +/-100 metres). In our prototype framework we have provided a simple penumbra model that defines the area around a *captured value* in which the *actual value* may lie. However, depending on the application and level of inaccuracy awareness needed, more complex models of error may be required. Taking this to the extreme, the current value for a context element may be described in terms of a probability graph where there is a whole continuous range of likely and not so likely current values.

In addition to considering the error of the context element, a way of representing it needs to be chosen, too. For example, with location we need to decide if it is a name/cell based system or a co-ordinate based system, and if its the latter then the number of dimensions (x, y or x, y, z), the shape (point, rectangle, polygon, cube, mass, etc.), the units of measurement (latitude and longitude, metres, etc.), etc. In the framework we experimented with using latitude and longitude as a universal form of

measurement, a common denominator of location types. However, this was not terribly successful as it ignores the fact that different types of location are suited to different situations. For example, a latitude, longitude, and altitude may well accurately define the location of my office but this information is practically useless for most purposes as there is no convenient method of finding out one's location within a building in such a form. An active badge cell-based system would be much more appropriate in such circumstances. However, for specifying the location of a giraffe in the middle of an African wilderness expanse a latitude and longitude measure would clearly be more useful, allowing us to locate the giraffe with a GPS receiver. Another aspect to this problem is that some location systems may be relative to a particular object rather than to a fixed point of origin on the planet. For example, if an active badge system were to be deployed on a cruise ship there would be no static mapping possible between the location cells and the latitude and longitude co-ordinates. Indeed, for most internal cruise ship applications there probably would not be the need for such a mapping as they would most likely require a ship-relative form of location system. Therefore, rather than trying to enforce a universal location format we believe it is better to use the one that is best for the task at hand and provide support for converting between different formats as and when necessary. This "convert-on-demand" approach can also be applied to the subtle differences in data formats, e.g. converting between the different datums or precisions of a location.

Leonhardt et al's location service [Leonhardt, Magee 1996 - #81] [Leonhardt and Magee 1996 - #80] attempts to integrate all forms of location through a series of cell-spaces. However, although co-ordinate data such as latitude and longitude from a GPS receiver can be integrated in his model, it does so by transforming an inherently unstable form of location (i.e. precise latitude and longitude measurements of moving objects) into an inherently stable cell-space model, resulting in a high processing overhead in updating/reconfiguring the cell space. It faces the flip side of the problem that we encountered in trying to establish a common-denominator location system using latitude and longitude (although there are more goals that Leonhardt's location service is trying to achieve than simply providing a universal location format).

Location is a particularly complex context element to represent, with many different forms and units of measurement. However, at least some of the issues faced with

location data formats are likely to be encountered in developing support for other context elements as well.

The issues in representing context that we have considered thus far are low-level ones concerned with the extraction and representation of individual values. Yet more issues are raised when considering higher-level modelling of composite forms of context. For example, the trigger condition of a stick-e note may consist of a number of different context elements. It contains a separate trigger context for each general context element type, with the individual context elements of each list ORed together and the resulting Boolean values for the general context element types ANDed together (except for trigger contexts with no context elements, which are simply discarded). Refer back to Figure 4 for a visual representation of an example trigger condition. Location context elements are particularly suited to this form of logical representation as the relationship between location context elements to which a note is attached must inherently be OR. It is pointless to have a trigger condition that specifies a location trigger context as two different places simultaneously – the stick-e note will simply never trigger as the current location of the user can obviously not be more than one value at a time. It is possible to have such scenarios when using pretend context elements where, as discussed earlier, the normal laws of physics need not apply. However, the 'impossible contexts' that pretend context elements present are only used in simulations of current context and are not permitted in the trigger condition of a stick-e note. That is, any combination of contexts can be used as the current pretend context, but the stick-e note must be attached to a real physically possible context. Another possibility for the same-place-at-once scenario could be if we consider the possibility of a composite object whose components can be dispersed. We sacrifice support for attaching stick-e notes to impossible contexts and for modelling composite current context (both of which we do not expect to be widely required in the majority of context-aware applications) in return for a simpler model.

With context elements have different semantics than that of location context elements. Most notably, it *is* possible to have two context elements match the current context at the same time. For example, to be with Helena *and* with Geraldina *and* with Vince. Therefore modelling the *With* trigger contexts as a simple OR relationship between context elements is not satisfactory as we may also want to represent an AND

relationship between these context elements. One solution is to extend the context hierarchy to include a *With-Group* context element, but this would really be only a work-around solution for an inadequate method of expressing the logical relationships. In addition to the AND operator, a NOT operator may also be useful. For example, a *With* trigger context that defines: with Vince AND with Helena AND NOT with Geraldina.

In summary, the trigger condition model works well for the simple location-based sample applications we have developed but it does limit the flexibility of the context that a stick-e note can be attached to. There is a trade-off between providing a simple model that is easily understood and a more sophisticated one that can represent more complex contexts but at the cost of being less immediately understandable. The current model places the emphasis on ease of understandability, which it does this very well: there is little or no ambiguity in what the context specification can mean. However, it does limit the complexity of contexts that can be expressed and is probably too simplistic for many applications. For example, it cannot represent "with Peter in his office or with David in his office", as a single context. However, providing a full Boolean context logic that can permit any logical combination of context elements will probably be an off-putting interface for the novice user. Perhaps the best way to address this problem in future work would be to provide two independent methods of specifying the context of a stick-e note, the current simple list arrangement by default or, for more sophisticated applications/users, the ability to specify the context in terms of a full Boolean logic.

2.5   Issues in Triggering

Many context-aware applications will utilise the available context information in a reactive manner. That is, they will adapt or enhance some aspect of their behaviour based on changes in their current context. The stick-e note framework enables such behaviour with its triggering model, where stored stick-e notes whose context matches that of the current environment will be triggered and delivered to clients who are interested in them. In effect it facilitates a contextual form of event driven programming where the context of the stick-e note determines when the event should occur and the content of the stick-e note determines the type of event or reaction. It is

entirely up to the client to create and utilise the latter, in the case of the electronic Post-It note the content was a textual note that was displayed on the screen when it was triggered. However, it could just as easily be some executable content that is run when triggered, e.g. an environmental control system that turns off the lights when a room becomes vacant.

Although useful in many applications, triggering is not the only way of utilising context. Some applications may be more interested in a continual monitoring of context, such as continuously showing the current value of a context element, e.g. the current time, temperature, or location. Triggering, and the stick-e note framework in general, also attempt to hide or abstract context information from the client and/or user. This is an incentive for many applications/users as they wish to concentrate on their application-specific functionality and data, with contextualisation provided transparently via the triggering mechanism. However, there are some tasks in which the application or user desire to be actually immersed in the context information, where triggering and stick-e notes in general may not be so helpful, e.g. an interactive electronic mapping program. In developing the stick-e note framework we have deliberately focused our efforts on supporting the reactive/adaptive type of applications, exploring how a triggering service can be operated and provided.

The trigger process that we developed for the framework prototype operates as anticipated but has highlighted a number of inefficiencies that could be addressed in the future. One of the main concerns is that a trigger checking cycle is quite slow in execution, especially on the resource-starved handheld computer. It is the extraction of context data from the physical sensor devices, in particular from a GPS receiver with the electronic Post-It note program, which is the primary cause of delay. Every time the framework needs to check if a stick-e note's location matches the current location, the attached GPS receiver is consulted. This is performed by the GPS sensor device class that listens in to the communications port that it believes the GPS receiver is attached to, watching out for the location information in the NMEA data [Bennett 2000 - #10] [NMEA 2001 - #92] that is intermittently broadcast by the GPS receiver. The time delay lies in the intermittent nature of data broadcast, where a period of one or two seconds can pass by before the next set of data is transmitted. Often the delay is much smaller than this, but, for example, with one or more location context elements in each

of a group of twenty or more stick-e notes, the delays can soon mount up.

Unfortunately this delay is out of our control: it is a predetermined setting of the GPS hardware that will be incurred irrespective of whether we model sensor devices in a hierarchy or as autonomous sensor monitors (as we proposed in the previous section). However, what *can* be enhanced is the framework's method and frequency of interaction with a sensor device. At present, the current location is retrieved every time a location context element is checked, ensuring that the most up-to-date location data as possible is used. However, the amount of time between the individual checks of stick-e notes in a matching process is very small (if we exclude the GPS data extraction time) and hence the location is unlikely to change by any significant amount in that short space of time. Therefore we could simply retrieve the location data once per triggering cycle and cache the value so that the checks could be performed using this cached location instead of extracting the location direct from the physical GPS receiver each time.

This caching method could be further enhanced using the proposed autonomous sensor monitors. Each monitor could keep a cache of the most recent location data (or whichever context element is being captured) and could pass this cached value to the `SeEnvironment` on demand. However, unlike a `SeEnvironment` based cache that would be updated each time the matching process was performed, the monitor's cache could be kept for an indefinite period without requiring an update. The monitor could employ a more advanced cache refresh strategy so that it is only updated when necessary, not at arbitrary periods defined by the trigger checking frequency. Clients may have a quality-of-service arrangement whereby the monitor will examine the GPS receiver just enough times to keep within the specified location granularity, e.g. to keep the location updated so that it is always within 50 metres of the actual GPS receiver's current reading. The monitors could also adjust their sampling rate so that if, for instance, they detect the speed of movement is increasing then they can increase their sampling rate of the GPS receiver in order to keep within the quality of service threshold, and vice versa. As the autonomous sensor monitors are constantly running in the background as independent processes, they could also schedule their access of physical sensor devices for times of low processor load to further reduce the bottleneck

surrounding the matching process. Caching can also help in estimating the current value of a sensor that is temporarily not working, e.g. a GPS receiver in a tunnel.

Through autonomous sensor monitors the process of capturing current context for use in trigger checking can be made much more efficient, but we also need to consider the broader perspective of how the to improve the trigger checking cycle as a whole. Currently a manager component in the framework is configured to perform a trigger checking cycle of a specified number of stick-e notes at specified intervals, i.e. it is a static timer driven process. If the current context or stick-e notes have not changed since the last trigger cycle then this is a redundant activity as we already know what stick-e notes will and will not trigger in such circumstances. More subtly it is also a redundant activity if the current context or stick-e notes have not changed *enough*, i.e. not enough to cause any differences in triggering. This redundancy is as a result of the trigger checking process being designed around a polling mechanism rather than an event driven one.

The trigger checker could be converted an event driven mechanism with the introduction of the autonomous sensor monitors. The `SeManage` could register an interest threshold in a context element, causing any autonomous sensor monitor producing such a context element to send a notification message if the threshold were exceeded. For example, consider a wildlife guide family of stick-e notes on an African game park visitor's GPS-enabled handheld computer. The `SeManage` may deduce that none of the stick-e notes is going to trigger until the user has reached the park gate. Therefore it could tell the GPS sensor monitor to not inform it of the location until within a certain distance of the entrance gate, and only then need it start trigger checking. Of course, the same general method could be applied on a much finer scale too, e.g. information on hippopotami does not need to be triggered until near the river.

An alternative strategy would be to calculate a "super-context" that is the union of all contexts that the user is likely to have in the next hour (or some other defined time period). The system could then trigger all the notes that match this super-context and use the resulting group of notes as a cache until the user strays out of the super-context, thus reducing the processing overhead for each trigger-checking cycle.

Having addressed the way in which changes in context are obtained and the manner in which the trigger checking is invoked, we now need to address the remaining aspect of trigger checking: the method in which a collection of stick-e notes is checked. The prototype framework simply cycles through the collection in the order in which they were opened. However, by applying some ordering to the stick-e notes, it should be possible to perform the process in a more efficient manner. For example, if the context element of primary importance is location (which is likely to indeed be the case for many applications that we envision) then the stick-e notes could be ordered by proximity to the current position. This can be visualised as lifting up a fishing net by a single knot in the centre and having the other knots (representing stick-e notes) cascade down from this central point. Given such an ordering we need only continue checking through the set of notes until reaching the first one that does not match the current context, as all notes after that are inherently more distant and hence not going to match either. Further work needs to be carried out to investigate whether the overhead involved in the sorting of stick-e notes necessary for this type of matching will be small enough to make it worthwhile doing, especially in large collections of notes. Further thought is also required on how ordering could be applied to many context elements simultaneously; perhaps by having several orderings of stick-e notes for each type of context element and then only checking the stick-e notes that are within range in every context element. Some context elements may not be sortable into any form useful for processing, e.g. names of animals can be sorted into alphabetical order but it doesn't really help in determining which is closest to matching the current *with* context element. Other more conventional techniques to improve triggering performance also need to be considered. For example, making use of database technology to extract a set of active stick-e notes by applying a query, which reflects the current context, to a relational database table, which represents the set of open stick-e notes. Further investigation is required into such possibilities to see if they are feasible, e.g. in the case of employing database technology to see if the varied, dynamic, and complex forms of context element data could be successfully represented and manipulated within a relational database. Essentially, these are all general problems of structuring and ordering in order to improve search speed.

Thus far we have considered triggering as a series of unconnected trigger-checking cycles. However, each trigger-checking cycle may need to know a little about the

previous cycles in order to determine what stick-e notes should trigger. For example, using the electronic Post-It note program I could create a stick-e note attached to Geraldina so that the next time I see her the Post-It program pops up the reminder I recorded to "Don't forget to ask Geraldina out for lunch". Whilst I am in the presence of Geraldina the stick-e note's trigger condition will continue to evaluate to true and each subsequent trigger checking cycle will result in the triggering of the note. However, it would clearly be inappropriate to display a new copy of the note each time it was triggered as I would soon end up swamped with hundreds of identical stick-e notes. We prevent such circumstances from occurring in the prototype framework through a selection of temporal restriction mechanisms that the author can apply to a stick-e note:

➤ **Re-trigger Timeout.** This is the simplest form of restriction, where a minimum duration between the triggering and re-triggering is specified. For example, a 20-minute re-trigger timeout would mean that, once triggered, the note could not be re-triggered until at least 20 minutes have elapsed. In the meantime the current context may have changed, preventing a re-triggering occurring directly after the re-inclusion of that stick-e note in the triggering process.

➤ **Specific Interval Restriction.** Prevents a stick-e note from triggering more than once in a given interval. The interval is expressed in terms of two bounding time and/or date stamps, e.g. 9:00 to 13:00 (which would prevent the stick-e note from triggering more than once during that period). The granularity of the interval can be extended to days, months or even years. Note, however, that a year interval will not be re-entered, unlike a time of day or day of week interval, e.g. a 9-12 interval could be re-used each day but a 1999-2000 interval will only be entered and exited once. Unlike the other two restriction mechanisms, it may be useful to specify multiple specific interval restrictions.

➤ **Generic Interval Restriction.** Similar to the previous type of restriction but instead of specifying the trigger restriction interval as two bounding time/date stamps it is defined as the current minute, hour, day, week, month or year. For example, specifying the restriction interval as the current day would prevent a

triggered stick-e note from re-triggering until tomorrow; specifying the specific day is not required.

In addition to restricting re-triggering there are some circumstances we have encountered in which the explicit de-triggering of stick-e notes is useful, which is currently not supported in our framework. To illustrate the need for de-triggering we can extend the previous example so that the invitation to lunch reminder is attached to the context of Geraldina's presence and also Helena's absence, i.e. to remind me to ask Geraldina out to lunch but only if Helena isn't around. In our current model if I was in the presence of Geraldina and not Helena then the reminder would, of course, be displayed. But if Helena were to enter the room a couple of seconds later the reminder would still be displayed even though I now do not wish it to be visible, i.e. it should be de-triggered and removed to prevent Helena from seeing it.

To support de-triggering the framework could maintain a list of active stick-e notes to which it could add any notes that are triggered. During each trigger checking cycle it could then remove any stick-e notes that no longer trigger and inform the clients who had received a copy of the note when it originally triggered that it now has been de-triggered. Depending on the application the client may chose to react to or ignore the message.

When employing triggering and de-triggering an interesting effect we call 'boundary hovering' can be encountered. For example, if I am on the boundary of what is deemed to be 'in the presence of Geraldina', and I start shuffling back and forth, then the stick-e note will be rapidly triggered and de-triggered as I appear to move in and out of that context. The problem is heightened if a temporal restriction on re-triggering is in place, in which case the reminder will be removed from my screen on my first shuffle and then not replaced until the temporal restriction has expired, which is clearly not what I want to happen. To avoid such problems a *de-trigger suspension period* could be enforced by the framework so that the stick-e note has to be consistently in a non-triggering state for a specified time period in order for it to de-trigger. This compensates for the border hovering effect as the elapsed time in a consistent non-triggering state will continually be reset if I am shuffling in and out of the context.

Although the temporal methods of restricting de-triggering and re-triggering are effective they require a good degree of judgement from the stick-e note author in order to define an appropriate amount of time. For example, exactly how long do I have to be out of Geraldina's presence before the stick-e note is de-triggered? It is not immediately obvious. Having the success of the framework's re-triggering/de-triggering strategy so dependent upon the author's sensible choice of restrictions and time periods is not entirely satisfactory. A preferable mechanism that we have proposed in later work is to define separate contextual zones for triggering and de-triggering, based on the user entering and leaving the context rather than on different timer mechanisms. We present this concept in chapter eight.

One final aspect of the triggering to consider is its distribution. Given the very limited processor resources of many platforms that will wish to utilise the stick-e note framework, offloading the bulk of the processing onto another computer may be desirable. The model-view-controller structure of the framework makes this a relatively easy task as the client device can simply implement a view interface and receive triggered stick-e notes from the remote controller and model components (though of course the communications medium used to connect them will differ). Distribution issues are the focus of the work presented in chapter eight.

2.6   HCI Issues

Obtaining context, representing context, and triggering by context are all issues focused around the stick-e note application framework design. However, when developing context-aware software there are a whole host of issues centred around the user interface design too. Good HCI design is important to any software that is to be exposed to the end-user, but even more so in context-aware software. Unlike desktop computers that can afford to expect the devoted attention of the user, context-aware software is most often found on handheld, wearable, or ubiquitous computers where such expectations would be unrealistic. To be successful such computing devices must fit in with the user's place and task rather than vice versa, so user-interface design is a paramount concern. HCI issues are an important part of the ecology tools work described in the next chapter, so we will limit our discussion of HCI issues here to ones of direct concern to the stick-e note framework and Post-It note program.

Unlike more conventional software such as word processors or spreadsheets, in which the program is explicitly driven through commands invoked by the user, stick-e note applications such as the electronic Post-It note have a certain degree of autonomy. This is apparent to the user in the form of processes that start on their own (e.g. the trigger-checking process) and user-interactions that are initiated by the computer rather than the user (e.g. informing the user of stick-e note that has triggered). This autonomy raises the need to keep the user informed of the processes being run on their behalf (as they may noticeably affect the operation of the computer, e.g. to impede its performance) and the need to interrupt the user's task in a non-obtrusive fashion when requesting a dialogue. These issues are faced particularly in the feedback of sensor device status and in the triggering of stick-e notes.

When using the electronic Post-It note program there are many potential errors that could arise from the attached sensor devices. For example, the sensor's batteries may run out, the umbilical cable linking them to the computer may become loose or detached, the sensor device may automatically turn itself off, there may be difficulty in obtaining a reading (e.g. the GPS receiver may not be able to detect enough satellites in view to obtain a location fix, or the active-badge wearer may have wandered into an area with no active badge sensors), etc. The user needs to be aware when such errors occur; otherwise the notes they create will not be attached to the correct context. Although a facility is provided in the Post-It program to view the status of the sensor devices, it currently has to be explicitly invoked from a menu option. That is, the user is currently unaware of a sensor device's status, including any failures, until a direct inquiry is made about that sensor device. A preferable solution would be some form of monitoring function carried out as a continuous background activity that alerts the user to any failures as they arise. Perhaps this could be conveyed in the form of a peripheral on-screen status indicator which, although normally only in the periphery of the user's attention, is made attention grabbing in critical error conditions.

When an electronic Post-It note is triggered the user is informed via a message that is briefly flashed on the screen in conjunction with the sounding of an audible beep. This works well in informing the user of the immediate actions of the computer, i.e. in triggering the note. However, there is little feedback before or after the event to alert the user that a note may be about to trigger or why a note did trigger. Presenting stick-e

notes only as they trigger strictly limits the user's view of the stick-e note 'landscape'. For instance, consider stick-e notes of a primarily location-oriented nature. They will pop-up on the screen as the user wanders through the areas that have notes attached to them; however, the user is never sure when or where the next stick-e note is going to appear. This could result in an activity rather reminiscent of minesweeping, where the user wanders around trying to find out where the stick-e notes are.

A much better approach would be to provide some means of forewarning the user of notes that are in close proximity of triggering. In the location-based example one can easily envision the usefulness of a user-centred map with the surrounding stick-e notes projected onto it. Similar schemes for other context elements could also be devised, e.g. for temperature we could plot stick-e notes on different levels of a thermometer that also shows the current temperature.

Providing such 'early warning systems' allows the user to anticipate when notes are going to trigger and to better visualise the surrounding stick-e note landscape. In addition they could be used to enable the user to preview notes, e.g. previewing a stick-e note by selecting one of the note icons that are projected onto the map view. Given this extra knowledge about the stick-e note landscape combined with the previewing facility, the user would be able to adapt their behaviour in order to move into, or away from, a stick-e note's particular context.

The other aspect of keeping users informed about triggering stick-e notes is the feedback given *after* a triggering has occurred. Currently, the user is simply informed that the note has been triggered and is provided with a means to view it. However, it would often be helpful to the user if a brief explanation of why the stick-e note was triggered is made available. For example, to help stick-e note authors verify that they have attached the stick-e note to the right context and also to help user's understand what current context resulted in the triggering of the stick-e note.

In summary, a combination of feedback before and after triggering allows the user to see what stick-e notes are likely to trigger soon and, after a stick-e note has triggered, to see why that triggering occurred.

In addition to receiving triggered electronic Post-It notes, we expect the user to be actively involved in creating them too. This involves the user typing in the textual message, a trivial task, and also specifying the context to attach the message to, a not so trivial task. The Post-It note program helps in the latter task by providing current values for all context elements that it is able to, but the user still has to select which ones they want to use in attaching the note (and to specify any pretend contexts). This could be further improved through context profiles which could be created for different tasks, e.g. a 'ask-x' reminder profile may just consist of the *With* context element. Then when authoring a stick-e note the user need only select the context profile rather than manually configure the list of context elements provided.

The authoring process may very well only involve using current context values, e.g. attaching an electronic Post-It to my current location. However, it may also use non-current values, e.g. attaching a reminder note to Geraldina whilst she is not present, or attaching a shopping list note to the local corner shop from home. In the latter cases the user must enter the context information manually. Considering the location context element, the Post-It program specifies it in units of latitude and longitude; units that are difficult for people to correlate with their physical environment. By automatically extracting the location from a GPS receiver the Post-It note program shields the user from this underlying representation when attaching an electronic Post-It note to the current context. However, when specifying a non-current context, or defining a pretend context element, or viewing an explanation of triggering, the user will be forced into the unattractive prospect of working directly with latitude and longitude data. However, there are ways to avoid this.

Firstly, we could present latitude and longitude transparently in the form of a map display. A map provides an immediately understandable medium for expressing location, assuming that the design of the map is appropriate for the user, e.g. a tourist map would not be appropriate for a water mains engineer. To indicate a location to the computer the user could simply point at a position on the map, and similarly the computer can position a cursor on the map to indicate a location to the user. Maps also extend the possibilities for more easily expressing complex forms of location. These complex forms may take the shape of areas drawn around the perimeter of a location. For example, a user could trace around the perimeter of the Cathedral grounds to

define a Cathedral location context element. This tracing could be performed on the map or with the aid of a GPS receiver to specify key points that make up the surrounding polygonal area.

Secondly, considering the way in which people give directions to one another, the importance of place names becomes apparent, especially for landmarks and well-known locations. For example, it is easier to specify the Cathedral by name rather than point to it, delimit its area on a map, or to specify a series of latitude and longitudes. To deal with place names in the stick-e note framework a location name server would need to be developed that stored a set of names, each associated with a geographic point or area. This database of names could be built up gradually by assigning a name to each location element that is created. In the current implementation of the framework a naming facility has been provided so that context elements can be tagged with names, and we intend to utilise this facility in developing a name server in future versions. One potential problem with names lies in conflicts or misunderstandings, e.g. I meant Braga Cathedral not Canterbury Cathedral. However, perhaps some of these conflicts could be resolved through a use of the user's context. In the aforementioned example if the user was located in the city of Braga then it is most likely (and it is a matter of probabilities rather than absolutes) they were referring to Braga's cathedral. Generally it would be suitable to have an abstract naming mechanism that is context dependent, e.g. when referring to the cathedral the user's current location is used to determine the most appropriate choice.

The methods of identification by map and by name are complementary. The former allows the initial association of a name with a defined area on a map (hiding the internal low-level representation of geographic points). The name can then be used to quickly identify a location and conversely the user can point to an area on the map and have its name is extracted. Naming is useful to most types of context element, not just location context elements. For example, specifying 'afternoon' instead of the time range, specifying 'hot' instead of the temperature range, etc. Location context elements happen to be especially suited to naming because the underlying units of data (such as latitude and longitude) are often more difficult for the user to map onto their world than place names and maps are.

Thus far we have discussed HCI issues that are primarily software related. However,

the ergonomics of the hardware platform are also crucial to the success of the system as a whole. A problem that immediately arose with the Post-It note implementation was the amount of equipment that the user was required to carry around. For a simple location-oriented device they required the handheld computer with a cable linking it to a separate GPS receiver. Holding the computer in one hand and the GPS unit in the other makes it difficult to carry out any tasks on the computer other than passively observing stick-e notes being triggered. Built-in GPS receivers could remedy this problem or perhaps more realistically, at least in the short term, it is sufficient to have some form of wearable GPS antenna that allows the actual receiver unit to be stowed in a pocket leaving two hands free for operation of the handheld computer. Besides the sheer bulk of equipment and cables, another problem with adding additional sensor devices is the typical lack of many communication ports on handheld computers. However, technologies such as blue-tooth [Bluetooth-Consortium 2001 - #11] are beginning to make wireless communications possible with a number of nearby devices simultaneously.

With the electronic Post-It note application, the design of the physical handheld computer itself was also cause for concern. A clam-shell screen and keyboard design may be more suitable in circumstances where mobility is thought of in terms of being easily transportable to other stationary locations rather than in being used whilst on the move. The notepad-like design of other handheld computers, such as the PalmPilot (which was not on the market when the first prototype was designed and built), with pen-based ready-to-use interfaces, better facilitate immediate access to the computer whilst on the move. The pen interface also provides distinct advantages over keyboard-based input. Firstly, it is a form of input that the computer-novice user is naturally familiar with, also overcoming the problem of trying to use a condensed keyboard on a small device. Secondly, the pen provides a more effective way of interacting with a graphical user-interface, enabling a point-and-click mode of operation. This is especially useful in stick-e notes where, for example, an area may need to be defined on a map displayed on the screen - it is much easier to draw an area with the pen rather than using the cursor keys of a small keyboard. But however good the handheld computer design, its small size, which makes it so convenient in a mobile environment, also makes it unattractive for prolonged periods of stick-e note authoring due to those very same attributes. Therefore it would seem sensible to shift tasks that do not need to be

performed whilst mobile onto a 'companion' desktop computer, e.g. activities like organising notes into families, authoring note content, etc. The advantages of this strategy are two-fold. Firstly, the functionality of the handheld computer programs can be reduced so that only the mobile-dependent tasks are offered, allowing for a simplified and more intuitive user interface. Secondly, by shifting work that does not need to be completed whilst mobile onto a companion desktop computer we can provide a more comfortable environment in which to perform those tasks, both in terms of physical ergonomic comfort and the computing facilities available (e.g. bigger screen, faster processor, etc.). To this end we developed a desktop PC stick-e note management and test system, details of which are given in appendix B.

2.7 Summary

This chapter has both investigated and demonstrated the feasibility of employing context-awareness in software and supporting its development through an underlying application framework. The framework is based on a Post-It metaphor whereby digital resources can be placed in context and triggered when the user enters or approaches that context. In the creation of the framework and a trial context-ware application, a number of issues have been explored. Strategies for obtaining context information from sensor devices were investigated in addition to methods of representing the resulting data collected (and the need to keep these two areas separate). We proposed different methods of improving the performance of trigger checking and examined the phenomenon of border-hovering and techniques to compensate for it. Finally, we explored the HCI issues that arise as a result of the unconventional types of environment, hardware, software, and data that both user and computer are exposed to.

The work on context-aware tools for ecology fieldwork that is presented in the next chapter continues our exploration of context-aware applications where the electronic Post-It note left off. The fieldwork tools are more sophisticated and designed for 'real world' use in the field with a large user base that helped us to evaluate their usability. Although triggering was not to have such a prominent role in the work, the nature and use of context and HCI issues are very much at the forefront.

Chapter 3:
Ecology Fieldwork Tools

Whereas the previous chapter described our context-aware work at the supporting framework level, this chapter describes our work on developing context-aware tools at the end-user application level. The aim of this work is to explore some novel applications of context-awareness and to ascertain how useful and feasible stick-e notes, and, more importantly, context-aware tools in general, would be in assisting in some real-world activities.

We chose ecology fieldwork as our application domain with the aim of providing ecologists with some form of context-aware computer that could support their various fieldwork activities. This domain seemed well suited to our research for two reasons in particular:

➢ Mobility within their environment, and the observation of that environment, is inherent in the ecologist's fieldwork activities, i.e. a changing context is central to their work and hence an ideal place in which to deploy context-aware tools.

➢ Although *GISs* (Geographical Information Systems) and other supporting software packages abound in the desktop computing environment there is currently little practical computing support for ecologists whilst in the field.

The tools that we present in this chapter aim to rectify this situation by providing ecology fieldworkers, and also fieldworkers from other disciplines (e.g. geologists, maintenance workers, etc.), with context-aware computing support in the field. A large part of this work was carried out under the "Mobile Computing in a Fieldwork Environment" project [Pascoe and Ryan 2000 - #100], the aim of which has been to promote the use of existing mobile computing hardware in higher education fieldwork settings through the development of new software tools. The development of these fieldwork tools and our close collaboration with the ecologists has allowed us to

explore some interesting issues of context-aware applications and also to develop a computing solution that has enabled us to judge the usefulness and practicality of context-aware technology in a "real world" setting. This is especially valuable, as context-aware applications all too often are confined to the research laboratory rather than the real world.

In this chapter we first present our assessment of the general application needs of ecology fieldworkers, worked out in collaboration with some ecology researchers and students in the Isle of Mull, followed by a description of the tools that were developed. We describe the extensive use of these tools in a giraffe behavioural study in Kenya, and the issues provoked by this usage, before concluding with the exploration of a novel context-aware application for identifying individual rhinoceroses.

3.1 Investigating the Ecologist's Needs

A group of ecologists from Manchester Metropolitan University agreed to collaborate with us in the development of the fieldwork tools and invited us to attend a number of fieldwork expeditions. These expeditions enabled us to gain a better understanding of their work, to ascertain their requirements, to test out early prototypes, and to gather feedback and suggestions. Before our first such venture into the field we developed an initial prototype of what we envisioned a fieldwork tool to be. This first prototype allowed us to demonstrate to the ecologists the possibilities of such tools, which in turn helped in gathering more concrete feedback and suggestions from them as to what functionality they would need and/or like to have.

Based on our previous experiences in developing the electronic Post-It note program we switched development platforms from the MS-DOS based HP-200LX to the U.S. Robotics Pilot handheld computer (in subsequent incarnations called PalmPilot and Palm). The platform change was made because, compared to the HP-200LX, all aspects of the Pilot's ergonomics, operating system, and user interface seemed more suited to being used in the field (see Figure 8 for a photo of the physical device).

The only disadvantages of the platform were the strictly limited memory and processor resources. These disadvantages will probably always hold in the immediate future: it has been said that small portable devices will always have a tenth the power and memory of

a desktop computer. It was due to these constraints that we decided that rather than porting the generic stick-e note framework to the Pilot and using that as the application's foundation, we would instead develop the prototype fieldwork tool from scratch without any supporting framework. We would still use the stick-e note concept but would build the necessary features directly into the software, the desire for increased performance being the driving factor. The previously developed electronic Post-It note program, as described in chapter two, could afford to sacrifice performance as it was only intended as a proof of concept rather than as a fully usable tool. However, with the ecology fieldwork tools we wanted to create a practical tool that could be given to ecologists to use in order to evaluate the effectiveness of a context-aware computer; so optimum performance was essential.

Figure 8. The Pilot handheld computer.

We understood from the ecologists that the majority of their fieldwork consisted of observation and data collection activities. The main role of the fieldwork tool, then, would be to assist the user in recording observations of various aspects of their environment. In essence, capturing and recording different context elements: a pursuit that a context-aware computer is obviously well adapted to. We therefore envisioned the fieldwork tool as a kind of current context monitoring device that could record snapshots of the current context for later perusal. Given this emphasis on monitoring, the main screen of the prototype was designed from the perspective of providing a viewing mechanism onto the current context, as shown in the screenshot in Figure 9.

The selected context element to be shown on this main display.

Pretend values of the selected context element.

The current value of the selected context element.

Stored stick-e notes ordered by the selected context element.

Short-cut to the most recently triggered stick-e notes.

Sticke Pad ▼ All
View By: ▼ Location
Present Position:
    TA 86,000E 112,767N
Simulated Locations:
++
Nearest Sticke Notes:

New   Triggered

Figure 9. Annotated screenshot of prototype's main screen.

The main screen (Figure 9) displays a view onto a particular context element, which can be selected by the user from a list of all the possible context elements. In addition to the current value of the context element and any simulated values (also referred to as pretend values) added by the user, the display also shows a list of note icons that represent stored stick-e notes. These icons are ordered by proximity to the current value of the selected context element, the closest note being displayed leftmost and the more distant ones being displayed rightmost. In effect, the line of icons acts as an index onto the set of pre-recorded notes, ordered by the selected context element. Triggered stick-e notes are also accessible from this screen by either selecting the "Triggered" button to invoke a triggered note list, or through the short-cut icons to the right of this button, which represent the most recently triggered stick-e notes. The triggered notes portion of the display is independent of the selected context element; it will remain the same whichever element is selected because triggering is caused by the whole context rather than individual context elements.

3.1.1   Defining Context

Considering some hypothetical contexts that an ecologist may be interested in we began to think of context elements as being divided into two categories. Firstly, commonly used context elements that could be built into a data collection program (e.g. the current location) and secondly, activity specific context elements (e.g. the species of bird currently being observed) which, due to the sheer diversity of possibilities, would have to be defined at runtime. Values of the former type of context element may be automatically obtainable by communicating with a sensor device (e.g. communicating

with a GPS receiver to obtain the current location). However, the latter type of context element would have to be supplied by the user (e.g. the user may enter "Chough" to specify the type of bird currently being observed). In the prototype we initially built in support for location, time and temperature common context elements (although only location and time values could be automatically obtained) and provided a facility to represent application specific context elements in a general manner using simple numeric and textual data types.
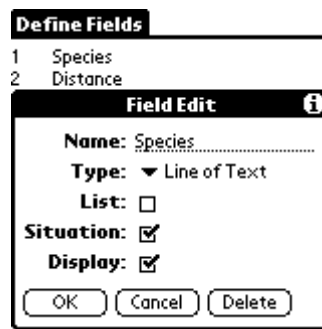
Figure 10. Editing one of the context element fields included in a stick-e note.

Specifying the sub-set of context (including common and application specific elements) of interest in a particular fieldwork activity is carried out by defining the set of context element fields to be recorded in a stick-e note, as illustrated in Figure 10.

A user-defined context element field is created for each unit of data that the ecologist is interested in collecting. These fields are created by specifying a name and a context element or simple data type to represent the value (currently limited to location, temperature, and time for common context elements, and number, line of text, and notepad for simple data types). The user can tailor the field further by indicating if it should be used in the trigger checking process and if it should be displayed when reviewing the note. This is useful as some context element fields may be used more for triggering and be of less interest when reviewing a note, whereas others may not be useful for triggering but do contain data of interest when reviewing a note. These are not mutually exclusive attributes, i.e. a context element field can be used for both triggering and reviewing. The attributes make the functioning of the program more efficient as they remove the need to check certain context elements in the triggering process and remove unwanted data from the screen when reviewing a note. In the user interface we represent these attributes with 'situation' and 'display' checkboxes. The

situation checkbox determines if the field forms part of the situation that the stick-e note is recording, i.e. if it is a field that the note can trigger on. The display checkbox specifies if the field value should be displayed if the stick-e note is triggered. Using a combination of these two attributes, fields can be defined that are:

➢ **Both situation and display**. Where the field forms part of the situation that is being recorded and is also displayed as part of the triggered stick-e note.

➢ **Just display.** Where the field is used to store information to display when the note triggers, in effect augmenting the situation.

➢ **Just situation.** Where the field forms part of the situation (so it is used in the trigger-checking process) but is not displayed in the triggered note.

➢ **Neither.** Where the data is recorded for later reference in a different program and perhaps on a different computer. Such fields, in our experience, are uncommon.

The field definitions can be added, removed or changed without affecting any of the previously recorded stick-e notes because each note keeps a copy of its field definitions.

The main display constructs its list of available context elements by combining the set of built in common context elements, which are always available, with the set specified in the context element field list that defines the structure of a stick-e note. Selecting one of the available context elements causes the main display to show that context element's current value, simulated values, and notes nearest the current value. The program maintains this information for each context element irrespective of whether it is an application specific one modelled by a simple data type or a common one represented by a real context element.

We originally intended to add a graphical view mode for the common context elements to support the visualisation of the current and simulated context element values and the distribution of stick-e notes around those values. This was not done, but we still believe it to be a useful feature. For example, for the location context element a graphical map view could be displayed showing the current value, simulated values, and location of recorded stick-e notes, all plotted onto the map display. Such visualisations would be limited to the built-in common context elements as the user-defined context elements

are represented by general-purpose numeric and textual data types that would be difficult to make a sensible visualisation of without more knowledge of the application specific data that they represent.

3.1.2   Creating A Stick-e Note

The user creates a new stick-e note by selecting the "New" button on the main display. This invokes a screen that presents the new stick-e note as an electronic form, where each field of the form represents a context element of the stick-e note. If available, the current value of each context element is inserted into the appropriate stick-e note field by default, although the user can manually edit each field, as shown in Figure 11. Note that we deliberately freeze the values for the fields after the note is created, i.e. changes to the current context are not reflected in the stick-e note after its creation. This is because the stick-e notes are being used to capture the current context at a particular point in time and not to continually monitor the context.

Only the *current value* of a context element is used to set the default value of a stick-e note field. Simulated values are not used because we view them primarily as a mechanism to retrieve and preview stick-e notes by pretending to be in a another context, i.e. they are only considered by the triggering process and not the note authoring process. Simulated values are more suited for use in triggering rather than authoring as there could be a number of simulated values for one context element, unlike the singular current value. Multiple values can be happily used in triggering (in effect the simulated values act as a retrieval query) but only a single value can be used as a default value for a stick-e note field.

Figure 11. Screen shots of (i) the new stick-e note form display, (ii) manually editing the distance field, and (iii) manually editing the location field.

In this first prototype the same form display is invoked to show a recorded stick-e note when clicking one of the nearest stick-e note icons from the main screen. However, we intended to replace that with a simple view screen that only shows the stick-e note fields that have been defined to be displayable. Extracting information from a GPS receiver and performing a triggering process was not implemented, and was merely simulated in the user interface (location data had to be transcribed manually where required). A large part of this first prototype was a user interface simulation of our initial vision of a fieldwork tool, with just enough functionality to test out its feasibility and usefulness to the fieldworker (i.e. the ability to record simple field observations) and to stimulate discussion of the ecologist's needs.

We tested out this early prototype on a MSc ecology course's two-week field trip to the Isle of Mull, Scotland. The aim was to get a better understanding of the ecology fieldworker's requirements and to see how well the prototype would perform the activities the fieldworkers were carrying out. We achieved this by shadowing the work of a selection of different students, trying to duplicate their paper-based in-field recordings on our prototype fieldwork tool. We describe two of the fieldwork projects we shadowed in the following sections.

3.1.3   Fieldwork Project #1: Knockvologan Bird Census

The aim of this student project was to re-sample the bird population of an area called Knockvologan, a remote moorland habitat that was censused two years ago. Since that original census the area had been fenced off and any farm animals removed, so the students were interested in evaluating the resulting regeneration of the natural environment from the perspective of differences in bird species density and diversity.

The new census was performed by revisiting the original census transect (a fieldwork term used to describe a well-defined path along which the ecologist walks to conduct their observations) and making new bird observations at the previous recording points. These observations commenced after a set period of arriving at the recording point in order to allow the wildlife to settle down after our arrival. During this calm down period the location, time, distance along the transect, and habitat description were recorded. Then for a duration of 5 minutes any bird heard or sighted was recorded, noting its species, if it was heard/seen/both, and its distance from the observation

point. Approximately twenty different bird observations would be recorded in a typical sampling period. These census points were revisited at different times of day throughout the two-week project period to ensure that birds which were active at different times of day were recorded.

The two ecology MSc students working on this project, Tim and Richard, conducted these fieldwork activities as per normal whilst we shadowed their activities, trying to duplicate their observation recording on the prototype fieldwork tool.

```
1. N 56° 17.577'     W 6° 19.851'
Time: 11:13-11:18 (all 5 minute durations)
Location: edge of estate
Distance: 0m
Habitat: open moorland

2. N 56° 17.638'     W 6° 19.888'
11:58
Near to ruined village
200m
Birch scrub on edge of moorland

3. N 56° 17.556'     W 6° 19.644'
13:00
200m
same as previous habitat

4. N 56° 17.528'     W 6° 19.528'
13:26
600m
same as previous habitat

5. N 56° 17.468'     W 6° 19.170'
14:00
Just over fence
1000m
bog + gorse scrub in front of birch wood.
```

Table 2. Sample of data recorded manually in the MemoPad application.

The information they recorded can be divided into two classes: the individual observations themselves (i.e. bird species, sighting/heard/both, and distance from observer) and also information common for one set of observations (i.e. location, time, distance along a transect, and habitat description). That is, information about an individual bird sighting and information about a whole session of sightings. When defining the stick-e note context element fields we only specified the ones relating to the individual bird sightings; otherwise the general session information would be

duplicated for each individual observation. The general session information was recorded separately in a memo of the Pilot's built-in MemoPad application, a sample of which is given in Table 2.

Note that in addition to the GPS derived position (which was manually entered into the Palm Pilot), a textual description of the location was also recorded in order to pinpoint it more accurately when trying to find the spot again in the future. The precise spot may be hard to find without this description due to the +/-100 metre error in an uncorrected GPS location fix.

To record the observation data a set of context element fields were created to define the stick-e note, see Table 3.

Name	Context Element / Simple Data Type	Situation (i.e. used when triggering)	Display (i.e. used when displaying)
Species	Line of text	✓	✓
Distance	Number	✓	✓
Call or Sighting	Line of text (using the letters 'c', 's', or 'b' to indicate call, sighting or both)	✓	✓
Location Ref.	Number (referencing the common location information entered into the MemoPad)	✓	✓
Misc	Notes (recording any other miscellaneous information for the observation, e.g. "not sure if it was a buzzard or an eagle")		✓

Table 3. The context element fields defined for a bird spotting stick-e note.

To initially set the system up as described took only a couple of minutes in which to define the stick-e note fields and to create a new MemoPad memo. This was easily carried out just before taking the first observations at the start of the first transect, i.e. it was a quick and simple process. In carrying out the fieldwork itself, the first activity on arriving at a new point along a transect was to record the general location information in a MemoPad memo. After the calm down period a new stick-e note would be created and completed for each individual bird sighting, as illustrated in Figure 12.

Figure 12. A stick-e note created for an individual bird sighting.

The stick-e note consisted solely of application specific context element fields; hence the current value of each element had to be manually entered as there are no automatic context element extraction mechanisms for application specific context elements.

3.1.4 Fieldwork Project #2: Bunessan Corvid & Eagle Census and Behavioural Study

Unlike the previous project, in which populations of all bird species were monitored to gauge environmental regeneration in an area, this project concentrated on ascertaining the populations of corvids (crows, rooks, jackdaws, etc.) and eagles only. The goal was to identify the distribution and population of these species and to observe their behaviour and interactions in different areas.

The census was carried out by walking a transect and performing observations of five minutes duration every 400 metres. As the area was already disturbed (we were walking along a populated, if sparsely so, road rather than the wilderness of the first project's census) the samples were taken immediately on arrival at the site rather than allowing a calm-down period first. The observations during the five minutes were more detailed, recording the bird species and timed behaviour and were accompanied by the weather, time, location and habitat data common to that 5 minute session. In addition to these five-minute observation sessions, any corvids or eagles spotted between observation points were also recorded but their behaviour was not monitored. The work was divided between two students, Sarah, who maintained a log of census-point observations, and Kathy, who maintained a log of the sightings made whilst walking the transect.

As with the first fieldwork project, the two ecology students conducted their work as per normal whilst we shadowed their activities trying to duplicate the data collection work on the Pilot.

What became immediately apparent was that there were two tasks to be completed: the stationary census-point observations and the observations noted whilst walking the transect. The data recorded varied slightly between the two: most notably a lot of textual notes about behaviour of the birds was recorded at a census point. In one instance two or three paper notebook pages (A5 size) of data were recorded whilst observing a kestrel for 20 minutes. Unfortunately the design of the prototype was such that only one of these tasks could be carried out because only one set of fields could be defined for a stick-e note. We chose to shadow the observations made whilst walking between observation points as this task seemed most different from the previous project.

Name	Context Element / Simple Data Type	Situation (i.e. used when triggering)	Display (i.e. used when displaying)
Species	Line of text	✓	✓
Location	Ordnance Survey Grid Reference (manually derived from GPS receiver)	✓	✓
Grid Reference	Ordnance Survey Grid Reference (manually derived from map)		
Distance	Number	✓	✓
Flight	Line of Text	✓	
Bearing	Line of Text	✓	✓
Time	Line of Text	✓	
Misc	Notes (recording any other miscellaneous information for this particular case, e.g. "not sure if it was a buzzard or an eagle")		✓

Table 4. The context element fields defined for a corvid/eagle observation stick-e note.

The observations recorded in the walking census were each distinct sessions in themselves, so there was no common session information to be recorded into the MemoPad as was the case in the first fieldwork project. All the data for an observation

could captured in a single stick-e note; the fields for which are presented in Table 4 and illustrated in action in the screenshots of Figure 11.

Note that the Grid Reference field is one of the rare cases in which a context element field is neither used in the triggering process nor in the display of the triggered note. This is because the field was used as a verification mechanism whereby after downloading the data to a desktop computer the locations derived from the GPS could be compared with the locations derived using conventional map-reading techniques. That is, after being recorded the field was only of further interest when downloaded to a desktop computer, not in the fieldwork tool itself.

3.1.5 Requirements of a Fieldwork Tool

As a result of the use of the prototype system in these two fieldwork projects, and after discussion with the ecologists, a number of general requirements were raised. We present them in this section.

3.1.5.1 To Automatically Capture as Much Context as Possible

The more context elements that can be gathered automatically the less data the ecologist has to manually enter. This is especially relevant to the location context element as the current value is already derived from a GPS receiver in the paper-based system but is subject to transcription errors when recording the current latitude and longitude from the GPS receiver's display. A simple connection between Pilot and GPS receiver (as we intended to provide in future prototypes) would both speed up the data collection process and eliminate transcription errors in recording the location information. A time context element would also be a useful (and simple to implement) addition for the ecology fieldwork activities as temporal data comprises a lot of their behavioural observation work (as was the case with the bird behavioural observations of the second project's 5 minute census points). Built-in support for capturing the time would eliminate the need to carry a stopwatch and to manually monitor and transcribe start and stop times of different behaviours.

A GPS receiver is now considered an essential piece of equipment in most ecology fieldwork. It is an extremely useful tool because the precise location is often difficult to

establish by manual methods due to homogenous terrain or lack of maps for an area. The location accuracy required is dependent upon the particular study. For some fieldwork the uncorrected GPS signal is satisfactory, for others post-processing to correct the signal is adequate, for others still, some form of real-time correction (e.g. differential GPS) is required.

In general, the ecology fieldworker would benefit greatly from a richer range of context element support. The same is also true of the simple data types that are used to represent application specific context elements that are not built-in to the software. Especially useful would be an enumerated type and some form of set type. The former would assist in modelling application specific context elements like bird species, colour, weather, etc. The latter would allow multiple values to be stored in a single field, e.g. a list of bird behaviours recorded in a single bird observation note. There *is* a list attribute provided in the prototype when defining a context element field, but currently it is not used.

3.1.5.2   *To Support Location and Time in Particular*

Both fieldwork projects were primarily data collection based activities, as are most ecology fieldwork projects. However, there is a lot of variation from project to project in what type of data is collected and the method in which it is collected. The two most common methods of collecting data are transect walking, and behavioural observations. The former records sightings of a species of interest whilst traversing a fixed path or at specified points along that path, perhaps with additional data such as a more detailed description of the species, its distance from the transect, its bearing, its current state, etc. The latter records changes in a species' activity over time, with the view to building up a time budget of the species' different behaviours. Both methods are inherently associated with a location and a time, although in the course of an observation it is generally the case that location is of primary importance to the transect method whereas time is of primary importance to the behavioural observation method.

Other methods of data collection, which we were not directly involved with in the first field trial, include quadrate sampling and mapping. In quadrate sampling an area is divided into quadrates in which various species are counted, giving a total for each quadrate and allowing a distribution of species to be calculated. Mapping is concerned

with identifying the location and boundaries of objects and areas, e.g. mapping a river, defining the area of a fire, etc. Location is important in both of the methods but in quadrate sampling the location is most likely an abstract quadrate number rather than a form of geographic measurement such as latitude and longitude that would typically be used in mapping. Utilising the fieldwork tools' location-awareness, new and improved sampling methods could also be more easily developed, e.g. random sampling, and complex multi-point/dynamic transect lines.

What all these data collection methods have in common is that they are heavily reliant on context information. All of the data collected can be considered part of the context, but location and time are especially important as they are often used as a form of index onto all the other context information. For example, the ecologist may record all the bird species along a transect and then later on may wish to find out what species are present in a particular location; as another example, an ecologist may record the activity of a crow and later on wish to see how much time it spent exhibiting a particular behaviour. Although each element of data recorded is considered a context element, some of them, like location and time, are of critical importance in order to make sense of the rest. Conveniently, time and location context elements are among the easiest to obtain automatically (through GPS receivers or internal system clocks) and could greatly enhance the usefulness of a context-aware fieldwork tool.

3.1.5.3   To Support Multiple Views of Context

The major problem we encountered with the prototype was its singular view of what context is. In designing the prototype we envisioned the ecologists collecting data by taking snap-shots of the current context, and provided a way to define what subset of the current context was currently of interest to them by allowing the addition and removal of context element fields from the stick-e note structure. When the ecologist had configured the stick-e note with the appropriate context element fields to match their interest, a snap-shot could be taken in the form of a stored stick-e note. However, what we had not considered is that the ecologist may have several different concurrent threads of interest. For example, in the bird census there was an interest in information about the context elements related to individual observations (e.g. species, distance, call or sighting, etc.) and those related to the observation session as a whole (e.g. habitat

type, location, time, etc.). It is difficult to model these distinct interests in the prototype as only one set of context element fields may be defined for the stick-e note. Changing all the fields each time the ecologist switches her interest is simply infeasible so in the trial we recorded the session data in the Pilot's MemoPad and just recorded the individual observations with our software. What is needed is the ability to define different types of stick-e note to capture different types of interest in the current context, and to be able to relate these stick-e notes with each other.

3.1.5.4   To Make Context-Awareness Transparent to the User and Task

Our inherent interest in context-awareness had also somewhat distorted our view of what would be relevant to display on the main screen. We had envisioned current context as being the most important piece of data and so had designed the main display primarily as a means to view the different context elements of the current context. A secondary role was to present the notes nearest to the currently displayed context element and those that had triggered. However, the ecologists were not particularly interested in viewing the current values of various context elements: they simply wanted to capture this information at intervals and be able to occasionally review the recorded notes. That is, what they really required is an interface oriented around the recording and retrieval of stick-e notes rather than the interface provided, which is oriented around displaying different elements of the current context.

In the fieldwork application it is more appropriate to have the context-awareness actively supporting the task at hand in a manner that is transparent to the user; otherwise it becomes a distraction or, worse, an interference, to their activities. Although it is easy to create stick-e notes, the current method of retrieving or reviewing them is cumbersome. The nearest note's or triggered note's icon list is intricate and time consuming to use at best, and provides no cue as to a note's content before it is opened. It is this stored note view that needs most improvement and enlargement as, in addition to creating stick-e notes, it is the centre of interest for the ecology fieldworker.

3.1.5.5   To Provide Dynamic Data Visualisations

The fieldwork computer is able to organise the collected data in a much more sophisticated and dynamic way than compared to a paper notebook. Facilities could be

provided that make further use of this ability in order to aid in the in-field management and review of stick-e notes, such as graphical views of different types of context element with stick-e notes plotted according to proximity. For example, in addition to the main screen's nearest note list we could also add a GIS-type view (in a slimmed down form) to display the current location of the user and the locations of the recorded stick-e notes on a geographic map.

3.1.5.6   To Support Stick-e Note Prototyping

The prototype's method of defining a stick-e note proved to be a good match for how the ecologists' data collection strategies are developed. When formulating their data collection strategy, although they have an approximate idea of what types of data they would like to record, it is only when out in the field that what is, and is not, possible to record becomes apparent. The data collected from observations therefore evolves out of a growing understanding of the subject over the initial period of their study. The prototype tool suits this evolutionary approach to establishing the data collection strategy by allowing context element fields to be added and removed from the stick-e note as required without affecting previously recorded stick-e notes in any way.

3.1.5.7   To Provide a Quick-to-Use Interface

It soon became apparent that the speed of operation of the prototype was critical. Observations in both fieldwork projects would often occur simultaneously, resulting in periods of intensive data recording. The software 'engine' was able to cope with these peaks of activity satisfactorily but the user interface became a bottleneck where it was quite difficult to enter the data sufficiently quickly. Some of the suggestions made for future prototypes, such as automatic context capturing, should help to ease the data input burden. However, the design of the user interface itself should also be investigated to see how high-volume rapid data entry could be better supported.

3.1.5.8   To Survive the Field

One general observation about the practicality of a context-aware field computer is the harsh environment to which it is subjected. Unlike the clean and relatively safe office environment typical of handheld computer use, we were using these devices in the

wilderness, where knocks, drops, mud, rain, sun and dust, all play their part in making the conditions rather inhospitable for such a device. Although our work concentrates on developing context-aware software, a rugged platform is still required on which to deliver it. The Pilot construction proved to be robust enough to survive the elements, for this first field trial at least.

Surviving the fieldwork environment is one thing, but actually being suited to it is another. The ecologists need a tool that is as unobtrusive as possible and that assists with the task at hand rather than distracting from it. For example, one could design a laptop to survive the harsh environment but it would be too big and bulky to be of much use to the ecologist. Again, out of all the available solutions we tried (including the Apple Newton, HP 200 LX, and Psion Series 3) the Pilot provided the most ergonomic solution for the field. Its compact and convenient notepad-like design and pen-driven interface made it easy to retrieve quickly from a pocket and immediately start recording notes. It was even possible to use it whilst simultaneously walking and talking.

3.1.5.9   *To Provide Links with the Desktop*

Capturing the observations on the fieldwork computer or paper notebook is just the start of the ecology project as a whole. To use the data to answer ecological questions it will be analysed by entering it into a statistical software package, mathematical model, or *GIS* (Geographical Information System), etc. Therefore an essential part of the fieldwork tool is to be able to download the collected stick-e notes to a desktop computer and be able to export it in a format suitable for such packages. A desktop computer could also act as a central repository for a group of ecologists where each person could contribute to a shared database of information on the desktop computer by downloading the stick-e notes from their fieldwork computer each day. For example, each ecologist could contribute some bird observation stick-e notes made during the day to a database of bird sightings on the desktop computer. This would enable ecologists to efficiently and effortlessly collate data gathered in a joint project and to allow the distribution of data to the appropriate people with shared interests. Being able to review the data almost immediately after collection would also allow the ecologists to adapt their fieldwork strategy in quick response to any deficiencies found.

3.1.5.10 To Ensure Data Security

Most ecology fieldwork projects are underpinned by data collected in fieldwork expeditions. Therefore the security of the data collected is paramount: it would not do to lose it all on the last day of the expedition. We believe that the fieldwork computer will be more secure than the current paper based system, as the data can be downloaded to a base computer each day and a back-up strategy for that base computer devised. Hence, the most data that could possibly be lost at any one time would be only one day's worth (due to some failure in the fieldwork computer before downloading). With a paper notebook system the data is much less secure (there are numerous ecologist "horror stories" of dropping notebooks in rivers, etc.).

3.1.5.11 To Offer a Field Guide Service

In addition to collecting data, the fieldwork tool could also perhaps be used as a context-aware field guide that works in a similar fashion to the electronic Post-It note: automatically displaying information of interest in context. The first two or three days of the student's fieldwork projects were spent introducing them to the environment and its various geological and ecological features of interest. If the fieldwork tool stored this information as a set of stick-e notes then they could be triggered and displayed to the user when approaching the associated sights. Perhaps stick-e notes collected by ecologists in earlier studies could also be brought out into the field by an ecologist updating the study or just interested in reviewing the area in which it was conducted. Other more general stick-e notes could contain ecology information that members of the public may be interested in when walking through an area. In addition to stick-e notes attached to locations they may also be attached to a specific animal if, for example, radio-tracking equipment was connected to the Pilot so that any radio-tagged animals in the vicinity could be automatically detected. In summary, a stick-e note field guide would be similar to having a number of ecology experts offering information appropriate to the current situation.

3.1.6 Summary

Our first prototype gave us the opportunity to test out our initial ideas of the form and function of a fieldwork tool, and to gain a better understanding of the ecology fieldworker's needs. We worked with a group of MSc ecology students from the

Manchester Metropolitan University who were conducting various fieldwork projects, ranging from observational studies of deer to assessing populations of snails, during a one week field trip to the Isle of Mull. They had a variety of support for processing their collected data when back at the base-camp (for example, using a laptop computer with GIS or database packages) but had no practical computing support in the field; the prototype was a pioneering effort to provide them with such facilities.

We shadowed the data collection work of two groups of students with our initial fieldwork tool prototype and it quickly became apparent that the common ground of all fieldwork projects lay in data collection. Recording observations in the field with the prototype highlighted many requirements that we would address in future prototypes in our continuing efforts to assist the ecologist's fieldwork activities. The only requirement that we did not subsequently explore is the concept of a field guide service, which is a full-blown project in its own right.

3.2   Generic Fieldwork Tools for Data Collection

Informed by the ecologist's requirements derived from our experiments and experiences with the first prototype, we created a suite of fieldwork tools to assist in their data collection activities. We developed these tools in close collaboration with Kathy Pinkney, an ecologist who had kindly agreed to trial them in her two-month behavioural study of giraffe in Kenya (though, of course, the aim was to develop a general stick-e note application that could be widely used in many fieldwork activities, i.e. not one specifically designed for giraffe behavioural studies!). This enabled an assessment to be made of how useful the fully-functional fieldwork tools were in a real fieldwork project, under real fieldwork condition, and over an extended period of time. This second prototype (for the Kenyan giraffe study) contrasted with the first prototype (used with the ecology students on the Isle of Mull), where the ecologists still used their paper notebooks, and an additional observer (me) duplicated their activities on a computer.

3.2.1   The Changing Nature of a Stick-e Note

Our original concept of stick-e notes was as an electronic version of a Post-It note, albeit with more sophisticated content and context possibilities. These could aid in

fieldworkers' activities by allowing data to be recorded in the form of a note attached to a particular context, e.g. attaching a "saw rare orchid here" note to the orchid's location. In addition to automatically tagging a note to a context for reference purposes, the program could automatically 'pop-up' the note on the screen the next time the user's current context matched the one that the note is attached to. The notion of context (e.g. location) and content (e.g. orchid) being distinct and discrete entities is firmly entrenched in such an 'electronic tagging' model. However, is this distinction a valid one? Historically this differentiation was due to the attempt to create an electronic parallel of the Post-It note where there is obviously a separation of the notes written on the Post-It and the physical object to which it is attached. However, with the increased richness of context expressed in ecology fieldwork (e.g. times, temperatures, etc.) the distinction between context and content begins to blur and break down. For example, if taking a series of temperature readings at various locations around a volcano, is the temperature a context or is it content? It could in fact be considered as either or even both. Therefore modelling stick-e notes in terms of separate content and context seems to be increasingly unsatisfactory.

Our first prototype had addressed this concern by allowing a stick-e note to be specified in terms of a number of context element fields, each of which could be defined as part of the context to trigger on and/or part of the content to be displayed. The role of a stick-e note then, has changed from one of tagging to one of modelling a snapshot of the current context. There are an infinite number of possibilities of what context elements could be represented in a stick-e note, e.g. soil type, weather conditions, colour, heart rate of user, variety of apple, etc. In this new context-snapshot model of stick-e notes each of these are treated as equals, and the typical dominance of location as the context element to which other values are attached is eliminated, leaving a rich array of potential context elements to form a stick-e note.

Although a significant change in the structure of a stick-e note, it does not make the concept any less general purpose. A general context-aware framework based on this new notion of stick-e notes could be applied in just as many different application areas as the original concept, i.e. it is *not* specifically designed for ecological fieldwork. Considering a tourist guide application, it can still be implemented as a set of stick-e notes where each note will contain a set of context element fields. Some of those fields

will define the physically detectable parts of the context (i.e. the ones that will be considered as the context of the stick-e note for triggering purposes) and others will define information to supplement the physically detectable parts (i.e. the information that will be displayed to the user). For example, a simple guided tour could be constructed from a stick-e note of two context element fields: one for specifying the location and one for specifying a URL.

3.2.2   Hardware Employed

In the second prototype the U.S.Robotics PalmPilot (the successor to the Pilot used in the first prototype) linked to a Garmin 45 XL GPS receiver was used as the hardware platform (see Figure 13) and a laptop PC was used to backup the PalmPilot's stick-e notes.

Figure 13. U.S.Robotics PalmPilot linked to Garmin 45 XL GPS receiver.

3.2.3   A Suite of Three Programs

The software for the PalmPilot was designed and developed as three separate applications: the *StickePad* to create new stick-e notes, *StickePlates* to create new stick-e note templates, and the *StickeMap* to visualise stick-e notes containing location context elements via a map display. Although separate, the three programs work in harmony by operating on the same database of stick-e notes and by communicating with each other through a common area of storage. The reasons for the division are discussed in this section.

The user interface of the first prototype centred around one main screen where both current context and stick-e note information could be accessed. The central feature of the interface was the *view-by* control that allowed the user to select which context element was used to orient the view of the main screen. The current and simulated values of the selected context element were displayed and also a series of icons representing the stored stick-e notes, ordered according to their proximity with the current value of the context element.

Two of the main information elements of the screen, the current context value and the list of stick-e notes, were typically used independently of each other. For example, the user may want to find or check their current position, in which case the *current context value* element would be used, or to search for a pre-recorded stick-e note, in which case the *nearest stick-e notes* element would be used. Although convenient to have all these facilities in one place, this further reduces the already limited screen size for each information element. This results in a somewhat cramped display where the information is displayed in a small form to suit the screen rather than in the most accessible form to suit the user. For example, instead of the small textual readout of location (or other context element) it would be preferable to visualise this in the form of a map (or another context element visualisation) to make navigating easier. The nearest notes could also be plotted onto such a map display giving the user more information about where they are and the proximity of notes to their current location.

The main reason that the display of the first prototype took the form it did was that the nearest notes list was designed to serve two purposes: (1) to order stick-e notes by proximity and (2) to allow editing access to a list of pre-recorded notes. The result was a compromise that served neither task particularly well. To remedy this situation we decided to separate ordering by proximity and editing access into two separate displays. In fact, we went further and partitioned the displays into two different programs, so creating the StickePad and StickeMap.

The StickeMap provides a grid-based map onto which the user's current position is plotted along with that of any nearby stick-e notes. It provides information on current context and nearby stick-e notes in one display. Although these two pieces of information tend to be sought independently of each other (the user tends to seek the

current location or the nearest stick-e notes at different times) combining the two gives a better sense of the context of either by illustrating them in relation to one another.

Some simple map functionality is provided such as zooming and scaling. Also by clicking on a stick-e note icon the user is transported to an editing screen for that note, though for the most part the user will edit notes directly through the StickePad. Note that we could have supported a background image on the StickeMap to allow scanned paper maps to be utilised. However, as all of our field trials were to be carried out in areas with no available maps we chose not to implement this feature for the time being.

A stick-e note icon. Clicking on it launches the StickePad with that particular note automatically opened for editing.

Buttons to update the *cursor position* from a GPS reading, *re-center* the map on the user's current position, and also to *zoom* in or out.

StickeMap 2 of 3

CurPos Re-center + -

Shows the number of stick-e notes displayed out of the total number that have a location field.

Indicates the current position of the user. Note: the icon appears bolder if the GPS is attached and functioning.

Figure 14. Screenshot of the StickeMap program.

The StickePad is used for creating new stick-e notes and viewing and editing stored ones. Unlike the first prototype, where searching for a pre-recorded note was literally a hit and miss affair (clicking randomly through the line of nearest note icons), in the second prototype a little more information is given to the author to enable them to better hone in on the note that they desire. This information is presented in the form of a scrollable list of stick-e notes represented by their associated sequential number, time and date of creation, and, optionally, a summary of the note's context element fields.

A separate StickeMap application for visualising and manipulating the location context element could be perceived as going against the general philosophy of the updated stick-e note concept in which each context element field is considered equal to all the others, with no special treatment of one over the other. However, the principle of uniform treatment across all context element fields applies to their processing rather than their representation in the interface. Different types of context elements inherently have different display requirements, e.g. there are a lot of visualisation possibilities that

can help the user manipulate location data, but there is little that can be usefully done to visualise the number of giraffe except displaying the numeric value. The first prototype's *view-by* control was conceived out of the idea of treating context elements equally, which should indeed be the case for trigger-checking and other processes, but for viewing data the different context elements inherently require different approaches to visualisation.
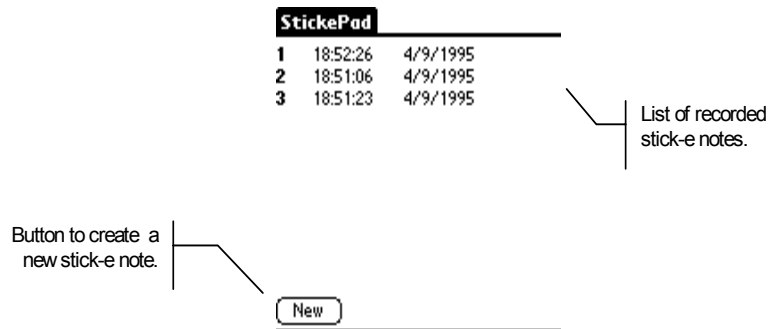
Figure 15. Screenshot of StickePad main screen.

When creating and editing stick-e notes, the StickeMap program is not required. The plain location data, in the form of latitude and longitude, can be worked with directly. The StickeMap is intended as a supplemental program to provide some useful context-aware features that focus on one particular context element, i.e. a map display for showing the location of the user and stick-e notes. We view the StickeMap as being similar to helper applications in Web browsers, providing a display service for a particular type of data, or, in the case of the fieldwork tool, a particular type of context element. In future work we hope to develop (and encourage others to develop) more helper applications for the fieldwork tool in order to support the visualisation and manipulation of other types contextual element.

In the first prototype we found that we required a more sophisticated method of defining the context element fields of a stick-e note. In particular the ecologists had interests in several distinct sets of context elements rather than in one single universal set. This was addressed in the second prototype by introducing the concept of stick-e note templates. A template enables the definition of a common set of context element fields from which stick-e notes can be constructed. This added a great deal more complexity to the StickePad user interface than the simple "define fields" menu item in the previous version. The creation of the templates is also a logically independent task

from that of creating stick-e notes. We therefore separated all the template creation and maintenance functionality into the third and final program, called StickePlates. This program allows the user to create a set of stick-e note templates and to define the set of context element fields that constitute each one, as illustrated in Figure 16.
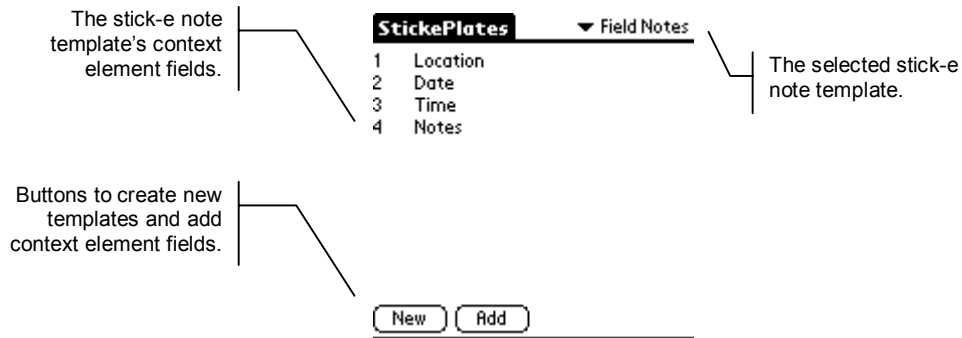
Figure 16. A template called "Field Notes" defined in the StickePlates program.

This separation left the user with a much simpler interface to the StickePad program where the infrequently-used template functions were removed, so avoiding any cluttering of the interface. In fact, the StickePlates program could even be removed from the PalmPilot altogether in some circumstances. This could be done to save space on the PalmPilot when using a standard or tried and tested set of templates that need no in-field modification. It can also be useful to remove the StickePlates program when equipping a group of people with a standard set of data collection templates: in this way they are prevented from modifying the templates in any way.

The rationale for separating the stick-e note software into the three separate programs, StickePad, StickePlates, and StickeMap, can be distilled into four main points:

➢ It reduces the apparent complexity of the small display by segmenting the program into functionally disparate areas, i.e. functions for defining a stick-e note template, creating and editing a new stick-e note, and visualising the collection of recorded stick-e notes (corresponding to the function of the StickePlates, StickePad, and StickeMap programs respectively).

➢ Separating the template definition into an independent program allows a data collection manager to define a standard set of templates which are given to the fieldworkers with the StickePad and StickeMap but, importantly, without the

StickePlates program. In this way the fieldworker is prevented from altering the template definitions.

➢ The spirit of stick-e note technology is to treat all context elements as equal and as such there should be the potential to visualise the stick-e note data by a variety of different context element views, not just a map for location context elements. By separating the location visualisation function from the main program a precedent is set whereby 'helper' visualisation programs based around other context elements may be developed and added to the system.

➢ The resource limitations of the PalmPilot on which the prototype was developed (and, indeed, other handheld hardware) favour a model whereby a selected group of small programs can be uploaded to perform a specific task rather than one monolithic program that is likely to include features which are redundant in some tasks.

In addition to improving the definition, editing, and visualisation of stick-e notes as a whole, the support for individual context elements and simple data types were also extended to include the following:

➢ **Bearing.** Specifies a direction in degrees 0 - 360. A range option is provided to allow a bearing range to be specified, e.g. from 67 to 92 degrees or even 350 to 10.

➢ **Boolean.** Provides an on/off, yes/no, or true/false checkbox.

➢ **Date.** A date or date-range, whose precision (e.g. dd+mm+yy, mm+yy, mm+dd, etc.) and format (e.g. USA or UK) can be specified.

➢ **List.** Facilitates the recording of any of the other supported types in a list (except for another list type). Provides the option of recording timing information with individual entries. The properties of the list element type may be adjusted just as if dealing with a single type.

➢ **Location.** Records location (currently only in the form of latitude and longitude).

➢ **Notepad.** Gives the user a display like the PalmPilot's built-in MemoPad application to record textual information.

➢ **Number.** An integer or real number, or number range, for which an upper and lower valid limit can be specified and a step increment defined, e.g. an integer that lies between 5 and 78 and which increments in steps of 4.

➢ **Time.** To record an actual time, a single duration of time, or a range of time, with a specified accuracy (hours, minutes, seconds).

Location, date, time and bearing provide context element support, whereas boolean, number, and notepad provide simple data type support enabling the representation of application specific context elements through generic types (similarly to the way a programming language allows data to be represented with generic types – though the range of base types required is larger). The remaining list type facilitates the storage of an undefined number of values for one context element, e.g. a list of people present in a room.

Simulated values and *triggered notes* were mostly omitted from the second prototype. Simulations were only implemented in the form of being able to simulate the current location in the StickeMap program in order to preview stick-e notes in an area different to the current one. The simulated location value was not used in the StickePad in creating stick-e notes, etc. (though the user could override the sensor input and enter a value manually; useful for sensor failure conditions), nor were simulations of other types of context element supported. The simulations feature was scaled back as it seemed likely that it would not be widely used by the ecologists in the field (though it may have uses later on in the desktop-based data analysis software). The trial of the first prototype indicated that most of the context element fields would be constantly changing from note to note rather than retaining a stable value (stable values are needed in order for simulated contexts to be useful in the automatic entry of data; otherwise there is no advantage over typing the new value directly into a new stick-e note's field).

The triggering mechanism was also omitted from the second prototype design. As with simulated values, the feature did not seem to be particularly useful to the majority of ecology fieldwork as the activities were based around the creation of stick-e notes rather than in their automatic retrieval (even in long running projects, whilst in the field the work generally focuses on recording new data and not inspecting previously recorded data). The concept of stick-e notes popping up on screen whilst roaming

through different contexts would be more attractive to tour guide applications that concentrate on *retrieving* data appropriate to the user's current context rather than in fieldwork data collection that concentrates on *storing* context information appropriate to the user's current task. A more suitable feature for the fieldwork environment is a means of gaining a new and perhaps broader perspective on the data that is being recorded, e.g. viewing the distribution of stick-e notes recorded around the user's current location via the StickeMap.

The emphasis of the second prototype is on exploiting context-awareness to capture, manipulate and visualise data (e.g. extracting GPS location data that can be used to fill in a stick-e note field or to view the user's current location on a map) rather than to use it as a way of automatically recalling information associated with the current context (i.e. triggering). This reflects the data collection nature of the work. However, rather than discarding the notions of triggering and simulations, the trial of the second prototype provided a good opportunity to consider how they could be exploited in such a setting, and how these concepts could be incorporated in future prototypes.

3.2.4   Using the Fieldwork Tools

Working with stick-e notes in the prototype is split into two distinct areas: definition and creation. This reflects the split of the software into the StickePlates program for defining and editing the stick-e note template, and the StickePad program for creating new notes based on a specific template and then allowing the user to edit them. The series of screen shots from the StickePlates program (see Table 5) demonstrates the process of defining a very simple stick-e note template.

1. The StickePlates template constructor. Note that the picklist in the upper right corner indicates that there are no templates defined yet.

2. Clicking *New* invokes the new template dialog.

3. After 'Oking' the previous dialog the picklist now shows the current template we're working on as 'giraffe'.

4. Clicking *Add* invokes a dialog to define a new field, where the type and name/label of the field can be entered.

5. Some field types have properties that can be edited by clicking the *Details* button.

6. After 'Oking' the field definition dialogs, the new field is added to the template.

Table 5. A sequence of StickePlates screenshots that illustrates the creation of a stick-e note template.

A number of stick-e note templates can be created and edited; the template picklist is used to select the current one. Having created a suitable stick-e note template, or set of them, the StickePad can then be used to create new stick-e notes. A very simple example to illustrate this process given in Table 6.
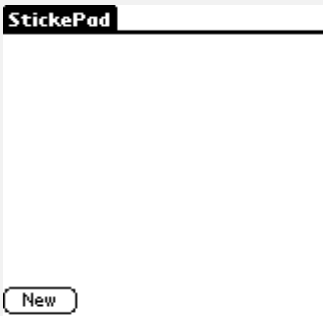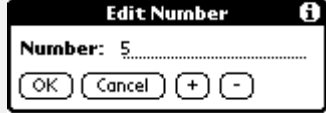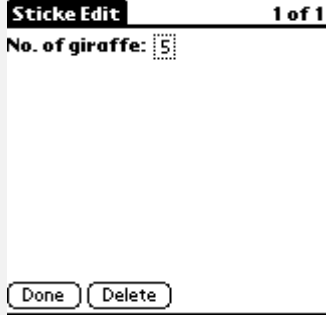
1. The StickePad; empty in the beginning.

2. Clicking the *New* button creates a new stick-e note that, by default, is generated from the last used or modified template. The template can be changed by selecting a different one from the picklist. Note: screen space is optimised in favour of the data boxes; the labels are never allowed to stretch beyond half the width of the screen.

3. Clicking on a data box *(No. of giraffe* in this example) invokes a dialog to edit the value of that stick-e note field. The + or - buttons in the *Edit Number* dialog increment or decrement the number by a step amount predefined in the stick-e note template.

StickePad

[New]

Sticke Edit   ▼ giraffe
No. of giraffe: [0]

[Done] [Delete]

Sticke Edit   ▼ giraffe
No. of giraffe: [0]

Edit Number   ℹ
Number: 5
[OK] [Cancel] [+] [-]

4. After editing a field, the respective value in the stick-e note is updated.

5. Clicking *Done* saves the changes to the stick-e note and it is added to the list of notes in the StickePad, annotated with the time and date of creation.

6. Clicking on the stick-e note entry to edit it invokes the same editing window as was used to create it. However, the template picklist is replaced with the sequence of the stick-e note within the collection - as changing the template is pointless at this stage because it would wipe out the recorded data and create a completely new stick-e note.

Sticke Edit   ▼ giraffe
No. of giraffe: [5]

[Done] [Delete]

StickePad
1   18:54:59   4/9/1995

[New]

Sticke Edit   1 of 1
No. of giraffe: [5]

[Done] [Delete]

Table 6. A sequence of StickePad screen-shots illustrating the creation of a new stick-e note from a template.
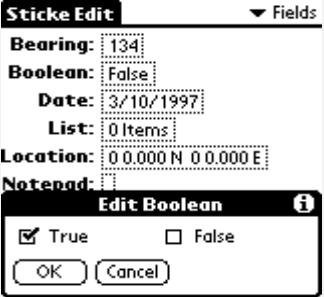
Having created some stick-e notes the user can either view them by scrolling through the list in the StickePad's main screen or by viewing the ones in the surrounding area on the StickeMap (see Figure 14).

The StickePad is where the focus of activity will be whilst the user is in the field, e.g. the ecologist recording their observations. The need to enter data into the StickePad in

a very rapid manner was highlighted in the fieldwork experiences of the first-generation prototype. Therefore, we have attempted to streamline its user interface and provide some time saving features.

Stick-e notes naturally exploit an awareness of context. The particular context elements available to the StickePad software are the location, from an attached GPS receiver, and the time and date, from the internal Pilot clock (although a more accurate time could potentially be extracted from the GPS receiver). Therefore, any fields of a location, date, or time type could automatically be filled in by the StickePad when a new stick-e note is created. It is likely that these will be the values that the user really wants to enter, as they will typically be collecting real-time and real-location data. However, any of the fields can be manually adjusted should the need arise.

For stick-e note fields that record application specific context elements through simple data types, the current value of the field cannot be automatically obtained by the StickePad. The best that can be achieved is to provide a helpful user interface to enter, view, and adjust this data. Indeed, such a philosophy is also usefully applicable to contexts whose values *are* automatically accessible. Some of the data entry interfaces are illustrated in Table 7.

The Stick-e Note	Bearing	Boolean
A stick-e note template called *Fields* was created to demonstrate the use of the various field types. The default values of the various types in a new form are shown here. *Date*, *Location* & *Time* are set to current values.	The bearing type allows the user to express a heading that lies between 0 and 359 degrees. To edit the figure one of the digits is clicked on and can be 'scrolled' up or down using the arrows. The dialog prevents an invalid value from ever being entered (as opposed to displaying a warning message when an incorrect value is entered).	The true or false values can be toggled using the check-boxes.
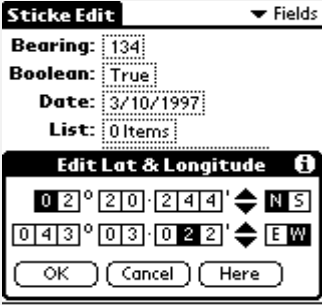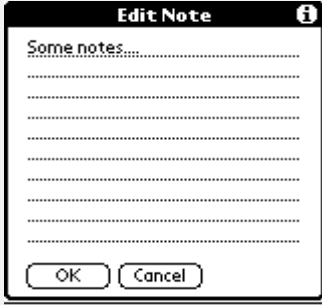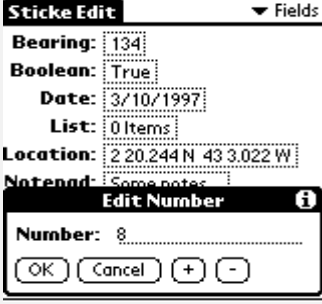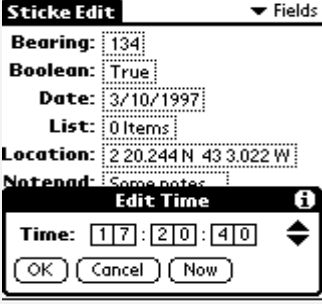
Date

The date dialog is re-used from the PalmPilot's built-in *Date Book* application, so its operation will be familiar with Pilot users.

Location

Much like the bearing type, this dialog allows the user to adjust the latitude & longitude (which would normally be obtained from a GPS receiver) and prevents the user from entering any erroneous values. The *Here* button is provided to allow the user to refresh the value from a connected GPS receiver.

NotePad

Textual notes can be entered in a dialog similar to the one used within the PalmPilot's built-in *NotePad* application.

Number

A number can be typed in directly or adjusted using the + or - buttons. These buttons increment or decrement the number respectively by the step value that is specified in the stick-e note template. Also, the two hardware scroll buttons of the PalmPilot have been overloaded to act in the same manner as the + or – buttons. As the user can easily locate and operate these buttons without looking at the device, this provides a useful counting mechanism that minimises the attention the user needs to devote to the screen.

Time

The time dialog follows in the tradition of the bearing and location dialogs by providing scrolling number boxes that enforce the entry of correct values. A *Now* button is provided to update the value with the current time.

List

The list dialog utilises the other type dialogs to create and edit values for each list element. Buttons are provided to add, delete or edit these elements. Optionally, elements can also be time-stamped upon their creation, as is shown here.

Table 7. A selection of screenshots of various context element editing dialogs.

For bearing, date, number, and time fields, a *range* option can be specified when defining the form template. When editing such a field the user is then presented with a

control in the bottom right-hand corner of the editing dialog to facilitate switching between the two values of the range.

3.2.5   A Giraffe Study Field Trial

As mentioned earlier, we were fortunate in having a willing volunteer to test out the second generation prototype: Kathy Pinkney, an MSc ecology student conducting a giraffe behavioural study at an ecology research centre in Sweetwaters Game Reserve near Nanyuki, Kenya. The main role of the reserve is to provide a sanctuary for black rhino and an environment that is conducive to an increasing population of these endangered animals. A number of studies, including the giraffe behavioural study, are being conducted to investigate how various elements of the reserve's ecosystem interact with and affect the rhino population, with the aim of using this data to aid in the management of an optimal environment for the rhino.

We have deployed the prototype tools in a number of other fieldwork projects including an archaeology expedition in southern Spain [Ryan, Pascoe 1997 - #119], a project assessing elephant impact on riverine areas, a study of zebra and gazelle behavioural interaction, and numerous other ecology projects. In this chapter we concentrate on the giraffe study in particular as it offered the most wide-ranging and intensive use of the prototype.

The ecological focus of the giraffe behavioural study was to gain a better understanding of the daily behaviour of giraffe and, in particular, their feeding habits and the resultant impact they have on the vegetation within the reserve. As anticipated, the majority of the work to be carried out in the field would be of a data-collection nature; the analysis of the data would be carried out when back in England. The computing research focus was to see how effectively out context-aware tools could be employed in a practical application and to identify future research issues from the needs and problems expressed in these practical experiences.

This trial was the first time that we put the prototype in the hands of the ecologists, and the first time that it would be used in collecting large amounts of non-trivial data over prolonged periods of time. The first few days in the reserve were spent seeing what would be possible to achieve in the giraffe study, both in terms of the giraffe study

itself and also how the prototype could be best employed. Kathy had an overall plan of what she would like to accomplish and the type of data she would like to collect. However, some first-hand knowledge and experience of the environment there was vital to establish the practical feasibility and constraints of the study. For example, through a process of trial and error we established the best way to observe giraffe and gauged the nearest distances at which they would tolerate us. In a similar experimental process, we investigated the best ways in which the data could be recorded in the prototype.

The process of designing a set of stick-e note templates was very much an evolutionary one. The initial templates were enhanced each day, based on our experiences of using them, in order to make the data entry process as easy as possible and to give the recorded data an understandable structure. Subtle innovations in the template design were made whilst out working in the field and more fundamental revisions made during the evenings back at the research centre. Three main criteria guided this template development process:

➢ *Ease and speed* of data entry.

➢ *Understandable template structures* so that recorded data could be easily retrieved and viewed.

➢ *Space optimisation* giving consideration to the limited memory of the Pilot.

Although minor modifications were still being made, a basic set of templates had been established at the end of the second week to service the four main data collection tasks: giraffe scan, focal, faecal and habitat observations. Each of these tasks is discussed in turn in the following.

3.2.5.1   Scan Observations

Giraffe were initially found by walking through the reserve until a group was encountered, at which point a scan survey would be conducted to identify various characteristics of the group. To record this information on the prototype a *scan* stick-e note template was defined with the fields illustrated in Table 8.

When creating a new scan stick-e note the *Date, Time* and *GPS* fields are all automatically completed by the StickePad program, which supplies the appropriate current values. Therefore, after creating the new stick-e note the ecologist could immediately begin observing the giraffes and counting their numbers rather than having to worry about establishing and recording the location or time of the observation.

Name	Context Element / Simple Data Type
Date	Date
Time	Time
GPS	Location
Total no.	Number
No. of males	Number
No. of females	Number
No. of sub-adults	Number
No. of juveniles	Number
Weather	NotePad
Misc. notes	NotePad

Table 8. The context element fields of a "Scan" stick-e note template.

3.2.5.2 *Focal Study*

Having completed the scan survey an individual giraffe would be singled out on which to conduct a more detailed focal study, observing all aspects of its behaviour but with particular attention paid to its feeding behaviour. This is where the vast majority of the data was collected. A *focal* template was constructed with the fields illustrated in Table 9 to record this information.

The first four fields describe the general characteristics about a specific giraffe. The behavioural observations are then all recorded in the *Behaviour* field, where for each individual unit of behaviour observed a new list element (in this case a *NotePad* type) is created to record it. The individual units of behaviour are composed of six main types: walking, walking and ruminating, standing, standing and ruminating, feeding, and out of sight. For all activities except feeding, only the activity type is recorded in the new *NotePad* list element, along with the time it occurred. The list type for the *Behaviour*

context element field was defined in the StickePlates program such that this timing data was automatically recorded, so only the activity type had to be entered manually. Even this had a degree of automation where instead of writing the full names of the behaviours each time, the PalmPilot's built-in shortcut mechanism was utilised. This allowed an abbreviation for a word or phrase to be entered that is automatically substituted by the PalmPilot for the full phrase, e.g. instead of typing 'walking and ruminating' only '^wr' need be input to generate the same text (where ^ represents the special PalmPilot shortcut symbol). Another time-saving feature employed was to reassign the four hardware application buttons on the Pilot (normally used to access the built-in Date Book, Address Book, To Do, and Memo applications) to provide a way of quickly swapping between the StickePlates, StickePad, and StickeMap programs.

Name	Context Element / Simple Data Type
Distance	Number
Direction	Bearing
Sex	NotePad
Age	Number
Behaviour	List of NotePads
Misc. Notes	NotePad

Table 9. The context element fields of a "Focal" stick-e note template.

The feeding activity is of particular interest to the giraffe study and consequently requires more detailed observation and data collection. A new feeding *NotePad* is created every time the giraffe starts browsing a different tree or moves to a new height on the same tree (measured to an accuracy of the nearest metre). When a giraffe starts feeding on a new tree a new *NotePad* list-element is created and the following details entered into it:

➢ The time the giraffe starts feeding (stamped automatically by the StickePad).

➢ A 'Feeding' activity type.

➢ Tree Species, e.g. Acacia, Uclea, etc.

➢ Height of browsing, e.g. 1, 2, 3, 4, 5 or 6 metres.

➤ Number of bites eaten at that height.

For subsequent recordings on the same tree (i.e. when the giraffe changes height) only the new height and number of bites are recorded in the new list element (the other values are assumed to be the same as the previous ones). This results in a list of *NotePad* elements such as those illustrated in Table 10.

The focal observations would typically last for ten to thirty minutes, very rarely lasting for longer than an hour. In a typical observation bout approximately 100 individual behaviour observations would be recorded. Recording this data was an intensive task where the giraffe could change feeding levels, etc., very quickly, requiring a lot of observation data to be entered in rapid succession. This data entry tended to come in spurts, where the giraffe would typically stand or walk for some time and then engage in a few minute's feeding activity again.

12:33:03	Feeding Acacia 2m 3
12:33:08	4m 12
12:33:25	Feeding Acacia 3m 2
12:33:31	Feeding Acacia 3m 5
12:33:32	2m 18
12:34:04	3m 4
12:34:15	Walking & Ruminating
12:35:34	Feeding Acacia 4m 10
12:35:59	Standing
…	…
12:56:43	Walking
12:57:23	Out of sight

Table 10. Example behavioural observations recorded in the behaviour list field.

3.2.5.3   *Habitat and Faecal Study*

Of all the data collection tasks involved in this study the focal observation of giraffe took top priority, as one has to make the most of the limited opportunities of observing the animals when they present themselves. However, once the giraffes have moved off

into the bush the relatively static data about the area in which they were present can be collected. This data consists of the diversity and density of vegetation in the area, recorded in a stick-e note derived from the template illustrated in Table 11.

Name	Context Element / Simple Data Type
Vegetation cover	Number
% Acacia	Number
% Uclea	Number
% Other	Number
Misc. notes	NotePad

Table 11. The context element fields of a "Habitat" stick-e note template.

The *vegetation cover* field expresses as a percentage the amount of the area that is covered with bushes or trees. Specific values are given for the Acacia and Uclea trees as these are the two main types of vegetation that are browsed by the giraffe. The ability to define an upper and lower range for a number in the template definition allows these fields to be checked for proper percentages, i.e. between 0 and 100. Also, defining a step amount of five allows the figures to be quickly toggled up and down using the + and – buttons or the overloaded hardware scroll buttons.

Name	Context Element / Simple Data Type
Sample no.	Number
GPS	Location
Date	Date
Condition	NotePad
Misc. notes	NotePad

Table 12. The context element fields of a "Faeces" stick-e note template.

In addition to studying the densities of particular plant species in the areas that giraffe have been observed feeding in, another method of determining their diet is by collecting faeces samples. This is a case where data is partly captured in the field, and partly as the result of later analysis. It is important to provide an easy way for the ecologist to link the two parts. These samples can then be later examined to try and identify the plants that the giraffe has consumed and subsequently excreted. The

following template provides a means of virtually tagging the samples to the location in which they were collected.

The giraffe faeces are placed in an envelope that is marked with a reference number that is used to associate the sample with the stick-e note (in which the number is also recorded). The date and location of the sample is recorded so that later on an attempt can be made to correlate giraffe sighted in the area with the faeces collected there. The condition of the faeces is also noted, being a useful indicator as to how long the sample has been lying there.

3.2.5.4   Giraffe Identification

Graphical facilities can also be useful in the collection of data. The final area that the prototype was used for was in helping identify individual giraffes, where the patterns of spots of some giraffe provide a distinctive marking with which to recognise them in the future.

Trig - 5th patch up from leg

Figure 17. Recording identifiable marks of a giraffe in the DinkyPad program.

To record the pattern for such future reference, a drawing application (called 'DinkyPad') was used on PalmPilot to make a sketch of the marking and to assign a name to the giraffe (see Figure 17). When conducting a focal study one of these sketches can be made and then referenced from the focal stick-e note by adding the name of the giraffe to it.

3.2.5.5   Securing the Data

The prototype fieldwork tool was used by the ecologists from the very start of the study. The only time paper was used was in sketching out a few ideas for the design of

the stick-e note templates and not at all in the data collection itself. Therefore, ensuring the safety of the electronic data collected was essential as it underpinned the whole project, and if lost or corrupted the whole time-consuming observational study would have to be carried out again. To ensure that such a situation did not arise, a backup strategy was devised that would keep three independent copies of the data secured in different locals:

➤ **Laptop.** Firstly, a copy of the data collected each day would be downloaded from the PalmPilot to a laptop computer at the research centre.

➤ **Disk.** Secondly, copies of all the stick-e note files were saved to disk each night just in case any problems arose with the laptop.

➤ **Email.** The final safety net was provided by the doctor's surgery in the nearest town that offered an email facility whereby we could send a ZIPed set of stick-e note backup files to a computer account back in England each week.

Even in a worst case scenario, where the PalmPilot is irreparably damaged in the field, only the current day's work would be lost, the rest being safely backed-up. As regards the data collection for the rest of the day, the ecologist would then simply have to resort back to a pen and paper approach.

3.2.5.6   Handing Over the Prototype

Kathy's proficiency in using the stick-e note software rapidly increased during first two weeks of use, and our role in offering help and advice was increasingly diminished. By the middle of the third week she had more than enough competence to operate the equipment and resolve any problems herself. This encompassed not only an understanding of the functions of the software but also a familiarity with the general principles of the equipment too, e.g. being able to identify a faulty cable as the reason why a location field was not being automatically filled in with the current location. After my departure Kathy continued to use the equipment with success until the end of the study some two to three months later. During that time, in addition to recording over 6000 observations using the initial set of templates, she also constructed some new templates and refined the existing ones.

3.2.6   Research and Design Issues

The second prototype successfully addressed all of the requirements established from our first fieldwork experiences at Mull, except for the field guide proposal. It also raised many new issues that are addressed in this section.

3.2.6.1   *The Role of Databases, GISs and Stick-e Notes*

Database, GIS or stick-e note technologies could all be usefully employed in fieldwork activities. However, does the new stick-e note technology provide any useful new concepts or facilities that are not already incorporated in present GIS or database technologies? A definition of the aims of the three technologies would be helpful in answering this:

➢ **Database:** a mechanism that stores data in a structured form and facilitates the updating and querying of that data.

➢ **GIS:** A tool for analysing and visualising data associated with geographical location. Typically, statistical analysis will be performed on different layers of data that represent different features (e.g. vegetation, roads, rivers, etc.).

➢ **Stick-e Notes:** a tool that exploits context-awareness for capturing, recalling, and viewing information about the user's current environment.

The domain of the database is that of developing optimum storage models and retrieval mechanisms. A database could actually work in harmony with a stick-e note system, providing the underlying model in which the data is stored. Alternatively, the stick-e note could be stored in the form of an HTML page with a location tag specified in the page's meta-data section. That is to say, the underlying storage model, whatever it may be, is not what defines the essence of a stick-e note; it is merely a foundation with which to support it.

The role of stick-e note technology is to provide a uniform way of working with, or viewing, the subset of data associated with the user's current environment. In effect, stick-e notes could provide a means to operate on the "here & now" subset of data in a database (or other underlying storage model) of the world (or part of it). The stick-e note system automatically retrieves data relevant to the user's present context, aids the

user in automatically capturing data about the current context, and provides methods of visualising the data in the current context.

Geographical Information Systems are oriented towards analysing and visualising data that has already been collected. This visualisation and analysis is achieved by relating data to geographical locations. Stick-e note technology works over a much broader range of tasks and context elements. It spans the whole range of activities from data collection to visualisation, and uses a broad range of data about the user's current environment. On the other hand GIS uses solely location (and perhaps location over time) as a means of plotting and analysing data, stick-e notes can operate with any context element of the user's environment such as temperature, time, animals present, etc. It is unlikely that stick-e notes will provide better geographic analysis than a GIS, due to this diversity of context elements, but they will be applicable to a much broader range of application areas. A GIS could in fact be used to help visualise stick-e note data through a location-oriented perspective.

When working with primarily location-oriented tasks, GIS and stick-e notes could work in harmony. The GIS could provide the static base system whilst the companion handheld device (equipped with the stick-e note system) could let the user collect and view data in the field, exploiting the system's knowledge of current context (in this case, most likely location data automatically extracted from a GPS receiver). An important task of a GIS is to analyse correlations between data, e.g. correlating animal densities with vegetation damage over a particular area. It is likely that correlating data would be useful both in the field and back at the base GIS system. The GIS derived correlations would probably be of a more complex nature that underpin strategic decisions, e.g. if vegetation damage is too high due to increased animal densities around a dam in the river then a decision may be made to build another dam to relieve the pressure on this area. The stick-e note correlations out in the field would probably be of a simpler nature and used in immediate decisions, e.g. if all the samples of elephant dung taken thus far can be seen to be concentrated in one particular area then the sampling method may be adjusted.

Stick-e notes provide a general-purpose concept that can be applied in a diverse range of programs that potentially work with an even broader range of context elements. It is the stick-e note technology that provides the means with which these programs can

gain access and utilise information about the user's current environment. Inherently, the applications and tasks likely to benefit most from stick-e note technology are those whose operation could exploit some interaction with the user's environment. For example, a tourist guide could greatly benefit from exploiting location data, whereas a word-processor probably would not.

Exactly what is meant by the user's environment varies. For example, in a tourist guide application the main elements of interest in the environment are location and perhaps time of day; the location context can be used to inform the user of sites of interest that are nearby, and the time can be used to ascertain if an attraction is currently open. However, in an ecological data collection task perhaps the weather conditions, vegetation type, and, again, location may be of interest. The current environment, then, is not merely a predefined set of physical attributes such as location, but is a *task-dependent* set of context elements. With stick-e note technology, applications can be created to capture, recall, view or manipulate data about, or related to, the task-related subset of the user's current environment, e.g. creating a template for a data collection task that contains the appropriate fields to record relevant context elements of the user's environment.

In essence, the novelty of stick-e note systems lies in the concept of context-aware data capture, retrieval, and visualisation. Databases and GIS systems can be used as engines or assistants to stick-e note systems, but the focus on context-aware capture, retrieval, and visualisation is what defines a stick-e note system.

3.2.6.2 *From Pretend to Expected Contexts*

Although simple data types can be used to model application specific context elements that have no explicit representation in the prototype, the software obviously cannot automatically ascertain the current value of this element. A feature that was present in the first prototype, *pretend context elements*, was intended to provide a substitute for this auto-capturing facility. Pretend context elements allowed the user to manually enter a current value for a context element that was then used as a simulated current value. These pretend values could also be used for context elements that were normally established automatically, e.g. for pretending a current location when no GPS device was attached.

The feature was removed from the second prototype because the effort to adjust the values of different fields through a pretend context dialog every time they changed outweighed the utility that was gained (note that the user effort lies in a combination of the interface design and the nature of the task itself). In many of the example applications considered, very few pretend contexts remained at a constant value; most changed rapidly, e.g. for each giraffe observation completed the location and many other fields would differ. Even for the few values that did stay constant for longer than one stick-e note authoring cycle, having to flip to the pretend context elements' screen in order to change the values resulted in little or no time saving over simply typing the value directly into each individual stick-e note (though perhaps there are better ways to construct this user interface in this regard). However, experience working with the system in the Kenyan trial led to another concept of how pretend contexts could be implemented that could lessen the burden of their definition and maintenance whilst enhancing the utility that they offer. Rather than simply specifying a single pretend value for a context element, the proposed system is based on the concept of having a selection of methods that can establish an *expected* value for a context element. Each context element field of a stick-e note is assigned one of these methods for predicting the next value. Some proposed methods for establishing the expected value (encompassing the pretend value and automatic extraction methods currently employed by the prototype) are listed here:

➢ **By Static Value.** This is the original pretend context method of operation, whereby the user manually enters the current value of a context element. This value is automatically inserted into any appropriate new stick-e note field. By default this will be the selected method for establishing an expected value.

➢ **By Adjusted Value.** This is an enhanced version of the static value method where, should the user change the value that was automatically inserted into the stick-e note field, then the pretend value is replaced with this newly adjusted value. This eliminates the main problem of taking too much time to adjust pretend context elements. The user would no longer need to flip to another dialog to adjust the pretend value; it is updated as part of the process the user has to carry out anyway.

- ➢ **By Most Used Value.** This method works by treating the most commonly entered value as the current pretend context value. For example if I have an 'Animal Spotted' field, and the most commonly observed animals have been giraffe, then 'Giraffe' will automatically be entered into the next note created.

- ➢ **By Predicted Value.** Some context elements are likely to have identifiable trends that can be extrapolated in order to predict what the next current value will be. For example, when walking a transect and taking samples every 100 metres, a detectable sequence of values for location will occur.

- ➢ **By Sequence.** Certain fields may be known at the outset to have a particular sequence to them, and these sequences could be specified in advance by the template designer. For example, if recording stick-e notes to tag photographs to a location, the *photo number* field could be assigned an *increment* sequence pattern so it is automatically incremented for each new photograph.

- ➢ **By Average Value.** For numerical based data, this attempts to estimate the new current context value by calculating the average of those collected so far.

- ➢ **Automatic.** This is used for fields where the handheld computer can automatically establish the current context element's value, e.g. by consulting the GPS receiver to establish the current location.

- ➢ **Blank.** In some cases having values that automatically appear can actually waste time instead of saving it, e.g. if the user has to erase the current value before entering the new one. Therefore, if there is no pattern to establishing the expected value automatically the field can be left blank when the stick-e note is created.

The emphasis of this new concept is in providing a variety of methods to establish the *expected value* rather than simply providing two polarised options based on automatic extraction from a sensor or manually defining a static pretend value. Some of the suggested new methods exploit a knowledge of the history of the context values that have been recorded in previously created stick-e notes in order to calculate an expected value. Unlike the previous pretend context system, which was appended to the main program, the expected values system should be an integral part of specifying a template.

An expected value method would always be chosen for a field, even if it is the *blank* option.

The application of the expected values is focused on providing current values for different context elements with as much automation as possible. This is a much narrower focus than the original pretend values that were also used as a mechanism for previewing notes by allowing the user to pretend they were in a certain situation. Such a use led to the need to support multiple values of pretend contexts, e.g. to show stick-e notes for both Canterbury and Nanyuki. Separating the provision of current pretend values from the previewing of notes (new methods for which are discussed later) allows for a simplification (i.e. no multiple values) that makes possible the more powerful concept of expected values.

3.2.6.3   The Need for Extensible and Flexible Context Element Support

The more that the stick-e note system understands about a particular context element, the greater utility it can offer to the user in terms of manipulation and simulation of that element. For example, treating location as a context element type, rather than as a simple type such as a set of numbers, allows specialist mechanisms to be developed, such as a map-based visualisation/manipulation tool. It also offers the potential of predicting future values of the context element with more accuracy, e.g. rather than looking for a sequence in a group of arbitrary numbers, we can look for a pattern in direction and speed. However, developing special type support for all context elements is simply not possible or practical both in terms of the sheer development effort required and the monolithic proportions that the stick-e note system would reach in trying to incorporate context elements ranging from giraffe behaviour to lava flow rate.

Currently, the stick-e note system provides support for a core set of types, which includes some specialist context element types and some general purpose simple data types. A specialist context element, such as location, date, or time, can incorporate tailor made element-specific abilities and features, e.g. location data can be extracted direct from a GPS receiver and plotted onto a map display. The general set of simple data types have no context-specific nature and hence can be used to represent a variety of different context elements, e.g. using the number type to represent temperature,

rainfall in millimetres, giraffe height, etc. However, these general types have no higher level features for capture, visualisation, etc.

Given the fact that different applications will have widely differing context element requirements that cannot be met with a single solution, what is required is some form of customisability in the stick-e note system beyond what is currently offered with simple data types. One way of providing this would be through an open interface to the stick-e note system into which context element modules could be slotted. These context element modules would offer various context-specific services to the stick-e note system in manipulating, capturing, viewing, and simulating a context element. They would also be physically separate code/programs/DLLs/applets/etc. from the main stick-e note system and could therefore be developed and distributed independently, akin to the development of plug-in modules for Web browsers. Using such a customisable model the stick-e note system could be tailored to the needs of an application so that only the modules needed for the task at hand are supplied, also having the beneficial side-effect of providing a solution that does not waste any valuable resources. It would also provide the ability to create very specialised applications that embedded more logic into the context element module, e.g. a *Rhinoceros* module that supplied the location of the nearest rhinoceros. There may be a requirement to support inter-module communication for cases such as the last example where services could be re-used, e.g. the *Rhinoceros* module could use the *location* module to ascertain the current location and then use this to calculate the nearest rhinoceros.

The current simple data types supported could be bundled into a general types module (as the number, notepad, etc. types are commonly used and don't take up much space) and then individual context element modules provided for each of the specialist context element types such as location, time, date, etc. If a particular application demanded the use of a specialist context element that was not included as standard then a new context element module could be developed and slotted into the stick-e note system.

The combination of the expected value and context element module concepts re-emphasises the role of stick-e notes as a tool for exploiting a knowledge of the user's current environment, providing support for working with the various context elements therein.

3.2.6.4 New Models of Triggering

The original stick-e note model employed triggering as a means to provide an automatic method of retrieving data that is related to the user's current environment. Triggering was conceived as a simple matching process where any stick-e notes that matched the user's current context were *triggered*, informing the user, by some undefined means, of the stick-e notes' active state. For example, in the electronic Post-It note prototype, when a user approached the Häagen-Dazs ice-cream shop a "You are near the ice-cream shop" note may be displayed on the user's handheld computer. This 'match and display' triggering model was conceived in the context of a stick-e note model that was primarily designed for tagging information to location context elements. Since that time the needs and concepts of the stick-e notes have changed, and the triggering model needs to be updated and extended to accommodate these changes.

Compared with the original location-tagged model of stick-e notes the fieldwork tools have a much larger array of context elements that can potentially act as triggers to information, as all fields of the stick-e note are now considered to be context elements. However, it is likely that a user may wish to select only a subset of context elements to act as triggers for stick-e notes. For example, the user may wish to just use the location context element so that all stick-e notes near their current location will trigger despite various other context elements in the stick-e notes not matching the current environment. In addition, rather than triggering stick-e notes that exactly match the user's current conditions it is likely that there will be an area around the user's current environment that is of interest, e.g. a temperature range of 30-60 degrees.

Although triggering was not developed further in the prototype, we carefully considered how it could be usefully employed in the Kenyan ecology work. We found that, rather than a single triggering mechanism, there are several potential methods of retrieving stick-e notes in context. Each of these methods are examined in the following sub-sections.

3.2.6.4.1 Passive Retrieval: Viewing Stick-e Notes in the Immediate Area

The first use of triggering that we consider is *passive retrieval*. This utilises the user's current context as a key with which to retrieve the set of stick-e notes that is most likely to be of immediate interest based on their contextual proximity. We call this "passive

retrieval" as it is a process that can be continually performed in the background without any participation required from the user, i.e. the user's role is a passive one, simply receiving triggered stick-e notes.

Perhaps the triggering model could be modified so that instead of dispatching a copy of a triggered stick-e note to interested clients, the state of a stick-e note could be stored within the note itself, which clients could then query or be informed of when it changes. In such a model it would be possible to make the nature of triggering more subtle and sophisticated by introducing different levels of trigger-state. Rather than a simple Boolean active/inactive value, perhaps there could be a 'Hot', 'Warm', 'Cool', and 'Cold' state indicating the nearness of the note to the user. Different context element fields of the stick-e note could be at different trigger-states, and parameters for each type of context element could be set to specify the threshold levels for these states. For example, the hot state of a location context could be a value within five metres of the current location, whereas the hot state of temperature maybe a value within two degrees of the current temperature. Some context elements would still have a simple active/inactive state if they had no continuous range of values (e.g. the type of animal), and some context elements may abstain from a trigger-state altogether. Allowing a context element to have no trigger-state effectively eliminates it from being used as a means to establish if the stick-e note it belongs to should trigger, which is useful when the user wants to define a stick-e note's 'triggerability' in terms of a particular subset of context element fields within the note. An average of the trigger-states of all the context element fields of a stick-e note could provide the overall trigger-state for the note.

Rather than popping up a message to display the triggered stick-e notes on screen, a more unobtrusive method of indicating triggering is required to match the subtler model of trigger-state levels. In this regard the current context element visualisations (such as the location context element's map display) can be enhanced so as to integrate indicators of the stick-e note's trigger-state. For example, the stick-e note icons displayed in the StickeMap could be changed or animated to indicate the different trigger-states. A discrete indicator bar could be included on some screens to provide quick access to the 'hottest' notes. Plotting the notes onto a map or other context element view also lets the user perceive the distribution of stick-e notes across a

particular context element, e.g. showing the spread, concentrations, etc., of notes on a map. This is most useful as it is often the case that the user is more interested in what lies ahead and around them than what is in the current spot. The potential disadvantage of this method if other tasks are being worked on on the PDA in the meantime is that the context visualisations take up much more space on the screen of the device.

3.2.6.4.2   *Active Retrieval: Previewing, Searching and Filtering*

As opposed to passive retrieval, *active retrieval* is a form of triggering in which the user does actively participate in order to preview, search for, or filter stick-e notes based on an explicitly specified interest rather than the current context. In the original stick-e note framework this took the form of pretend context elements that were used in trigger processing so that users could preview stick-e notes in a particular area by entering a number of pretend context element values. The triggering engine would treat the set of pretend context elements as current context element values and hence cause the triggering of any stick-e notes that matched these values, e.g. by pretending to be in Canterbury and Nanyuki stick-e notes with location context elements that described either town would be triggered.

Rather than 'accidentally' discovering stick-e notes through a passive retrieval mechanism, active retrieval calls for a more comprehensive search mechanism. This is probably more suited to a query mechanism rather than a pretend context model, as the user may wish to specify a set of stick-e notes described by matching multiple context elements, values, and ranges of values. For example, an ecologist may wish to find all the stick-e notes for giraffe observations where the weather has been sunny, the user has been within a two-mile radius of Pelican Dam or Elephant Bridge, and the observation was between 14:00 and 16:00. The resulting set of stick-e notes could then be listed or plotted onto any of the context element visualisations.

In addition to providing a method for searching the collection of stick-e notes, these database-like queries could also be used as a filter mechanism. For example, if the user were only interested in giraffe observation notes a simple query such as "animal = giraffe" could be used as a filter so that only the giraffe notes are active and displayed; notes about other animal observations are restricted by the filter, remain inactive, and are not displayed. As well as constraining the set of active stick-e notes on the handheld

computer, filtering will also be very useful on the desktop computer to allow the user to specify what subset of stick-e notes they wish to take out into the field.

These methods are essentially employing conventional database / information retrieval techniques to the collection of stick-e notes.

Thus far we have talked about previewing and searching as mechanisms to find stick-e notes from their context. Another important aspect of searching is to find the actual physical context from the stick-e note that is attached to it. Some form of navigation screen would be desirable that would help the user locate the stick-e note within their current environment, e.g. by informing the user how far away the note is and in what direction, or informing them of the temperature difference, etc. For example, an ecologist may want to re-examine and update information about some elephant dung by finding the location of a stick-e note that was previously created to describe it.

3.2.6.4.3   Alerts

The previous methods of passive and active retrieval work well for stick-e notes that the user knows about or for ones that the user wants to find out about. However, there are some stick-e notes that the user may not know of but which they should be actively alerted of. Some of these messages may be 'disposable' (e.g. a pager message that can be discarded once read) whereas others are permanently associated with certain conditions (e.g. a warning that you are approaching the firing range which is in use today).

3.2.6.4.4   Re-triggering and De-triggering

These were important issues in the original stick-e note framework discussed in chapter two because hovering around the boundary of a context element value could cause the same note to repeatedly pop-up, e.g. walking near the Cathedral perimeter may cause the "You are at the Cathedral" stick-e note to pop-up continually. The new triggering methods proposed reduce the impact of this boundary-hovering because instead of simply popping up stick-e notes that meet a set of conditions, the trigger-state of the stick-e notes are changed as the user approaches them and this gradually changing state is non-obtrusively made apparent to the user. However, this is only the case if the

context element values are relatively continuous and do not jump from one coarse value to another without warning, e.g. a hot to cold transition.

Re-triggering and de-triggering are still issues that need to be explored but the negative effects that they address are reduced in the proposed new model of triggering.

3.2.6.5   *Location Issues*

During the Kenyan fieldwork the current value of the location context element was extracted from a Garmin 45 XL GPS receiver, and the work has done before selective availability was removed from GPS. The value had no real-time or post-processed corrections applied in order to try and improve its accuracy, so the real value would, for most of the time, lie within a one hundred metre error radius of the supplied value. This proved to be more than adequate for the needs of the giraffe behavioural study, which took place over approximately a one hundred square kilometre area. The location data recorded was for use in either virtually tagging stick-e notes to a location or to aid in finding an observation site at a later date. The uncorrected GPS location proved satisfactory for both of these tasks.

The maps of the reserve that were used by the ecologists had one kilometre grid squares scaled to approximately two by two centimetres, which made plotting anything more accurate than one hundred metres impractical anyway. In addition, much of the map data had been collected through uncorrected GPS track logs, so the maps themselves were only accurate to within one hundred metres. These maps were based on the ARC 1960 datum with a UTM grid (zone 37N/37M). The prototype software supported only latitude and longitude but fortunately the UTM format was only required after downloading the data to a desktop computer (for plotting it onto one of the maps in a GIS) so it could therefore be converted after downloading.

Support for the plethora of potential location formats could certainly be improved, but how should this be accomplished? Resources in terms of processing and storage on the PalmPilot are limited and perhaps not well suited to the rigours of converting the latitude and longitude values transmitted by the GPS receiver into other formats. Providing this functionality on the desktop computer may be a better solution as it is free of such resource constraints. Perhaps when transferring the stick-e notes from the

PalmPilot to the desktop computer the location fields could be automatically translated into the format that the user desires. The majority of use of this transformed location data will occur on the desktop computer anyway so it seems logical to perform the conversions there also.

In the field it is useful to be able to locate your current position and the points of previous observations. The StickeMap allows the user to do this very easily, although currently the program does not display any additional information besides the grid, user's location, and stick-e notes' locations. This could be improved by providing a background map facility where the StickeMap program could overlay its current screen onto a bitmap image of a local map. The map image could be tagged with a latitude and longitude marker on the desktop computer before being transferred to the PalmPilot, enabling the StickeMap to appropriately scale it to the correct latitude and longitude grid. A selection of different maps could be provided on the desktop computer (e.g. for vegetation, roads, pathways, watersheds, aerial photos, etc.) from which the user could select the appropriate one for the day's activities. Alternatively, perhaps the user could also annotate the map with icons or sketches in order to depict landmarks or reference points, etc. Icons in particular would require much less processing time, e.g. when zooming in or out of the map the icon need not be resized or rescaled like a bitmap background would have to be. Such location-annotation icons could be implemented as a simple stick-e note consisting of a location, icon definition, and description field (so that when the user clicked on the icon a description could be displayed). Indeed, icons could be incorporated as a general feature of stick-e notes, where any note could have an icon field. Using these icon fields a specific icon could be created to represent a particular note in the map display or in any of the other context element visualisations.

Closely related to location is direction. In the Kenyan fieldwork, direction was commonly recorded, e.g. the direction of a group of giraffe, the direction of travel, etc. Currently the user needs to carry a compass in order to establish direction, but there are other methods that can be used to automatically estimate direction, although not always to the accuracy of a conventional compass. The first method uses a knowledge of the user's location so that a general direction of travel can be estimated if the user is keeping to a relatively constant heading (by simply projecting forward the user's general trajectory). The second method could provide a measure of direction that is

independent of the direction of travel by calculating it from the time and relative bearing of the sun. Finally, an electronic compass could be attached to the handheld computer to supply the direction in which the computer is facing. Any of the aforementioned methods could be most conveniently supplied as one of the proposed context element modules, extending the base stick-e note systems' context element support.

3.2.6.6   User Interface Issues

A number of helpful user interface features of the stick-e note software made the process of entering data much easier and faster than a conventional paper-notebook approach. Firstly, all the timing data of behavioural observations was automatically recorded by the stick-e note system, which tagged each behaviour change entered with the current time, eliminating the need to carry, operate and transcribe data from a stop watch. A counting feature was also provided that allowed numeric fields to be incremented by using two of the PalmPilot's hardware buttons, therefore not requiring the observer to look away from their telescope (and possibly lose the subject from the field of view) and also eliminating the need for a manual counter/clicker device. Finally, the PalmPilot's shortcut mechanism was used whereby a character, or short sequence of characters, can be mapped to a word or phrase, therefore reducing the amount the user has to write.

Recording data on single giraffes in the focal study (see Figure 18) proved to be the most stressful test of the stick-e note system, both in terms of the amount of data entered and the speed of entry. Stick-e notes derived from other templates could be easily completed with no immediate time limitations. In the focal study, however, data had to be entered quickly enough to keep up with the rapidly changing behaviour of the giraffe. Fortunately the aforementioned user interface features allowed the user to perform this activity faster and more efficiently that would have been possible with a paper notebook. According to the ecologist using the system, recording the same data in a manual system rather than in our prototype fieldwork tool would have been difficult to accomplish without a second person to assist in making timings and writing notes.

Figure 18. Kathy testing the tools before beginning a giraffe observation session.

Another useful feature was the provision of buttons in the field-editing dialogs that update the current value of a particular field automatically where possible, e.g. the *Here* button on a location field editing dialog that inserts the current location value extracted from an attached GPS receiver. The *Here* button was especially useful in circumstances where the GPS receiver had not calculated a location fix when a stick-e note was first created, allowing the user to update the field with the correct current location.

3.2.6.7   *Hardware Issues*

There were a number of hardware issues that were resolved during the first week. Firstly, the adequate robustness of the device was proven, after it suffered a number of falls and worked without fault during long periods of exposure to the strong midday sun and the dusty environment. Secondly, the physical ergonomics of the device were found to be well suited to the environment. The small and lightweight PalmPilot and GPS receiver were easily stowed in separate pockets whilst still being linked by the data cable, so adding little burden to Kathy in terms of additional equipment to carry around and also being immediately accessible when needed. Thirdly, the 'notepad' configuration of the device and Graffiti pen-based data entry method were also successful. The Graffiti hand-writing recognition system was quickly learnt by Kathy in a matter of minutes, and after approximately 2 days of occasional usage she was working at a speed similar to a traditional pen and paper equivalent, with few errors.

The notepad form-factor of the Pilot was a natural transition from a paper notebook method of recording data, and proved to be an intuitive way to enter data via a pen.

3.2.6.8   Grouping Stick-e Notes

The data that is recorded in each element of a behaviour list actually consists of several distinct data items, e.g. 'Feeding', 'Acacia', '4m', and '10'. However, as the list could only support a single context element type for each list item, the multiple values were recorded into a single NotePad type. This is not a satisfactory long-term solution because all the facilities that the other type-dialogs provide are unavailable when merely entering textual notes, e.g. the counting mechanism employing the Pilot's hardware buttons would have been ideally suited to recording the number of bites that a giraffe was taking; but instead the user has to maintain a mental note of the bites and then write down the total number at the end. It also makes it harder to download the data to the desktop computer in a meaningful form as there is no formal structure to the notes. In some way, support needs to be added to facilitate recording sets of data in a list.

A related issue is that of grouping separate stick-e notes together. For the giraffe observation work a set of stick-e notes was created for each encounter with a group of giraffe, one for the scan study, one or more for the focal study(s), and one for the habitat survey. As is often the case when working with large collections of data, methods of aggregating and structuring it are required, e.g. rather than three stick-e note icons shown on the map for different parts of the same group study, one group icon could be displayed instead. A common occurrence in the giraffe study was to update a previously recorded scan stick-e note if more giraffes emerged from the bush to join the group later on. If the stick-e notes were grouped into an observation set, the previous scan note should be much easier to find than in the current method of looking back through a list of all the recorded stick-e notes. Grouping also offers the possibility of having common fields that could be shared across all stick-e notes within the group, and hence eliminate any duplication of data.

The problems of structuring elements in the list and grouping stick-e notes together are closely related. One deals with grouping whole stick-e notes together at a macro level, the other with grouping individual elements of stick-e notes together at a micro level. The fact that the giraffe study stick-e note templates sit between these two extremes is

as a result of the ecologist trying to formulate the design at a level that makes accessing and viewing the contents most efficient. In an alternative design all the context element fields in a set of scan, focal, and habitat stick-e notes could be recorded in a single stick-e note if desired. However, in such a design accessing particular parts of the stick-e note (e.g. the habitat survey) is less convenient because the user has to scroll through the rest of the fields to find it, and the logical groupings of the scan, focal, and habitat data are less apparent. Similarly, if stick-e notes were created at a micro level, then the relationships between data would be lost and the user overwhelmed with a multitude of stick-e notes. The level at which the giraffe stick-e note templates encapsulate context element fields reflects how the data is perceived by the ecologist in the most commonly performed tasks, i.e. they most commonly work on the level of completing a scan, focal, or habitat survey. Therefore it is easier for them if the related sets of data for these tasks are grouped together. However, for less commonly performed tasks, such as looking for a previously recorded survey or editing individual behaviour entries, grouping of stick-e note data at different levels is required, such as grouping a set of notes together or grouping a set of fields together.

3.2.7 Summary

In a prolonged trial in an ecology fieldwork project in Kenya the prototype tools not only provided an electronic counterpart to the fieldworker's conventional paper notepad, but also provided many features that improved upon it in terms of both usability and speed. Automatic entry of the current values of location, date and time into the appropriate fields of a new stick-e note and the time-stamping of elements in a list provided critical time-savings; moreover features such as the overloaded hardware buttons, which allowed the incrementing or decrementing of numbers without the need to look at the device, provided an enhancement that made recording data whilst looking through the telescope much easier. Some features provided new functionality altogether, such as being able to immediately see on a map where all the stick-e notes are relative to the ecologist's current location.

A good indicator of the success of the system is how well it was received by the users. Kathy, the ecologist who was the primary user of the prototype in the ecology field trial in Kenya, found the system to be very useful and believed it provided a great utility to

her work to the extent that it would have been difficult to collect the same amount and variety of data in a manual system without an assistant to help. In general, all the ecologists at the research centre were very interested in the technology even at this early stage. The main benefits they perceived were of saving time in data collection, being able to download the data to a desktop computer instead of transcribing it from paper, and also being able to create many backups of the data. It would appear that there is a gap in the ecological-computing market that complements desktop based applications such as GIS, which are useful for processing collected data, with tools that aid the user to collect, view, and update data whilst in the field. Stick-e note software on small handheld devices seems to provide an ideal solution for such a market. This is not solely due to the resource limitations of available mobile devices but also in the nature of the tasks that users wish to perform in and out of the field (i.e. generally data collection in the field and data analysis and strategic decisions back at the base).

In summary, the prototype fieldwork tools and their trial in Kenya have shown that the development of stick-e note programs on handheld computers is possible, that they are useful, that they can be dramatically better than manual data collection methods, and that there is a user demand for them.

3.3   Esoteric Fieldwork Tools for Rhino Identification

The data collection tools described thus far provide the general means with which to design, create, and visualise stick-e notes. More esoteric functionality can be provided by creating companion applications to this basic tool set. To experiment with this idea we created a specialist application, called "RhinoIDer", that enabled an ecologist to identify and track an individual rhinoceros given some contextual cues. This application allowed us to extend our context-aware research into some very esoteric context elements and to explore some new methods of triggering.

Conventional identification of individual rhinos has only been possible through making visual sightings of an animal. However, it would also be useful to identify individuals from their footprints, as this would give the ecologists a considerably larger data set of specific rhinos linked to specific locations. It would also allow them to perform remote tracking of a rhino, where they could follow it from a much greater distance, even out

of sight, as they would be able to identify it from its tracks rather than needing to make a close approach to visually identify it.

Ecologists have supposed that a footprint could be used to identify an individual rhino in a similar way as a fingerprint can be used to identify an individual person. However, no-one has attempted this in the field as it would simply be too unwieldy: one must carry around historical footprint data, make measurements of the footprints found, and perform some comparison calculations on each possible rhino in order to identify the particular individual. The rhino identification tool that we envisioned aimed to automate most of this process so that it could be quickly and effectively carried out in the field.

There are four context elements that we decided to utilise in the *RhinoIDer* program:

➤ **Location.** The black rhinoceros tends to live within a well defined home range (not the same as a territory; home ranges often overlap). Therefore given the location of the footprint we can filter out the likely and not-so-likely candidates for creating a footprint.

➤ **Footprint measurements.** The ecologists established six individual measurements to define the shape of the rhino footprint, as shown in Figure 19. These can be compared with all the previously collected footprints for each rhino.
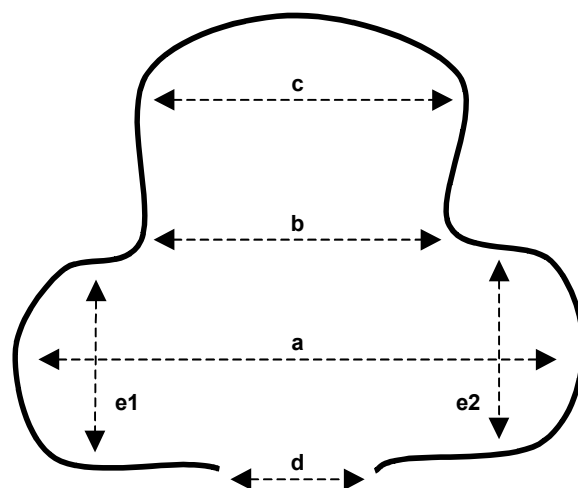
Figure 19. The 6 rhino footprint measurements.

➢ **Substrate type.** Whether it is thick mud or fine sand can have a great impact on the print that is made, so this needs to be considered in the comparison with historic footprint data.

➢ **Foot.** The front and rear feet can be significantly different, so we need to make the distinction and incorporate it in the comparison.

All four of these context elements utilised in rhino identification can be modelled in a footprint stick-e note, as shown in Table 13.

Name	Context Element / Simple Data Type
A	Number
B	Number
C	Number
D	Number
E1	Number
E2	Number
Substrate	Picklist of possible substrates
Foot	Picklist of 'rear' or 'front'
Location	Location
Misc. notes	NotePad

Table 13. A stick-e note template for a rhino footprint.

The historic data to which the footprint is compared can also be stored in stick-e notes, with one stick-e note per rhino, as shown in Table 14.

Name	Context Element / Simple Data Type
Name	Line of text
Footprint data	Footprint sub-notes
H.R. top left	Number
H.R. bottom right	Number
Misc. notes	NotePad

Table 14. A stick-e note for storing rhino information (including historic footprint data).

Note that the rhino information template utilises a new stick-e note nesting feature that allows a single or set of stick-e notes to be stored within the field of another stick-e note. In this case a number of footprint stick-e notes for a rhino are stored within its footprint data field. The "H.R. top left" and "H.R. bottom right" location fields define the home range of the rhino. Note that this is a rather crude measure of the home range (a polygon would be preferable) but it is sufficient for the purposes of this prototype.

Using these stick-e notes as its data model the RhinoIDer attempts to automate as much as possible of the four main tasks involved in the identification process:

➢ **Compilation of historic data.** Information about all the rhinos within the reserve (approximately 25) and previous footprint measurements can be stored as a set of stick-e notes. The set of prints can be transparently augmented as new footprints are discovered and recorded in the field.

➢ **Recording Measurements.** The location of the footprint is automatically ascertained and recorded using the attached GPS receiver. However, the individual footprint measurements, substrate type, and foot, are ascertained and entered manually. These could also potentially be automatically ascertained and recorded by utilising an attached digital camera and image analysis software. The resulting new types of context element could easily be incorporated into the stick-e note system if the suggested support for 'plug-in' context element modules was provided.

➢ **Comparison of data.** The RhinoIDer can rapidly compare an entered footprint with all other footprints for all possible rhinos.

➢ **Identification of the Rhino.** Ultimately the ecologist will make the final decision in identifying the individual rhino but the RhinoIDer can greatly assist by filtering out impossible candidates and prioritising the most likely ones. This process is effectively a form of triggering the matching or near-matching rhino stick-e notes.

The RhinoIDer relies on the StickePad to let the user create and complete new rhino footprint stick-e notes in the field. In this regard there is no difference to creating a stick-e note from any other template. However, should the ecologist want to identify the rhino to which the footprint belongs then they switch to the RhinoIDer program.

The RhinoIDer's main screen consists of two buttons labelled 'Trigger' and 'Confirm ID'. Selecting the trigger button will invoke the matching process in which a footprint stick-e note is compared with all the footprints stored for each of the rhinos. Each rhino that matches will be given a score based on the closeness of the match. This score is used to order the list of rhinos, as shown in Figure 20.
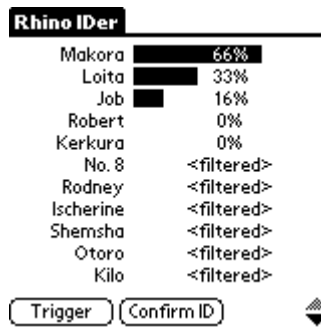
Figure 20. The RhinoIDer program finds Makora to be the best matching rhino, with Loita and Job a poor second and third place. Robert and Kerkura have a zero score, and the remaining rhinos have been filtered out due to a failed match on one or more critical fields.

The ecologist can examine the bar chart of rhino scores and may click on one of the entries to bring up the stick-e note containing that rhino's information. This could be considered as perusing a set of triggered stick-e notes. Once satisfied as to which rhino the footprint belongs to, the 'Confirm ID' button can be selected. This causes the score and name of one or more rhinos (more than one if the match is uncertain) to be tagged onto the end of the footprint stick-e note as separate fields. Although the footprint stick-e note could be automatically transferred to the footprint sub-note of the matching rhino at this point, so augmenting the rhino's historic footprint data, the ecologists preferred to retain control of this process themselves.

Figure 21. Defining the source of the footprint context to be the most recently recorded footprint stick-e note.

There are two sets of options that allow the ecologist to configure the matching and scoring process. The 'context sources' options, as shown in Figure 21, allow the ecologist to define the origin of the current footprint and location data. The footprint measurements can be derived from either the most recently recorded footprint stick-e note (default), a stick-e note of the user's selection, an average calculated from all footprints recorded so far, or no source specified. "No source specified" may be used if the ecologist is only interested in matching using location data. In turn, the source of the location data can be defined as coming from the GPS receiver, the footprint source (default), or no source.

Context Fields			
Name	**Score**	**Error**	**Filter**
a	2	10	
b	2	10	
c	2	10	
d	2	10	
e1	2	5	
e2	2	5	
Substrate	0	n/a	×
Foot	0	n/a	×
Location	10	n/a	

Done

Figure 22. Defining the role of each context field in the RhinoIDer.

The 'context fields' options, as illustrated in Figure 22, enable the ecologist to precisely define the contribution and effect that each context field has on the matching and scoring process. There are three attributes that can be defined for each element:

➤ **Score.** The ecologist can give each field a different score that will be added to the total score for a rhino if the particular field matches. This allows a weighting or confidence to be expressed as to the importance or accuracy of each different field.

➤ **Error.** For the numeric fields (i.e. footprint measurements) an error margin can be defined within which the compared value can lie but still be considered to match. Although the rhino's foot remains constant it is unlikely that any two footprints from the same rhino will exactly match to the millimetre accuracy in which they are measured. As with GPS derived location data, a degree of error must be allowed for.

➢ **Filter.** If the match of a particular field is essential to the match of the footprint as a whole then it may be defined as a filter field. If a filter field does not match then the matching process will discard the whole rhino stick-e note irrespective of the closeness of match of the other fields. Location is a particularly good field to filter on as it removes any rhinos that could not possibly have been in the area to create the footprint in the first place; the dimensions of the footprint in such a case are irrelevant.

Compared with the triggering model of our early Post-it note application, described in the stick-e notes chapter, the matching process here is much more sophisticated. Not only is an error 'penumbra' defined but also weightings given to particular context elements and the opportunity to filter by context elements that do not match. These new characteristics are born as a result of the need for a more subtle form of triggering, as earlier advocated. In this application the user is more interested in the degree of triggerability and the reasons for triggering rather than in a simplistic triggering/not triggering state (the reason is currently expressed as the percentage match but this could be expanded to show the individual components of the calculation). The application also requires a history of triggering to be maintained, where footprint stick-e notes are annotated with the rhinos with which they were associated. Even the origin of the current context is more sophisticated, being potentially derived from a number of different sources rather than simply the next location broadcast by a GPS receiver.

The RhinoIDer is now being used by the ecologists in Sweetwaters Game Reserve in Kenya. Although it still remains to be seen how successful footprint identification in general proves to be (a large database of footprint data first needs to be built up) the program itself has been working successfully in the field and allows for a good deal of experimentation in using different fields, weightings, etc. to determine a match. More importantly for us, the RhinoIDer illustrates how other programs can be added to the core set of stick-e note design, creation, and visualisation tools. It also demonstrates the use of a number of different types of context element and a more sophisticated form of stick-e note triggering in order to suggest the identity of a rhino based on certain contextual cues.

3.4   Summary

The aim of the work presented in this chapter was two-fold: firstly, to explore some novel applications of context-aware software, and secondly, to assess the usefulness and practicality of such applications in "real world" environments. The novel applications we created were aimed at the ecological fieldwork application domain, allowing users to perform general data collection and visualisation tasks in the field. We also provided a more esoteric tool that aided ecologists in recognising individual rhinos from their footprints. Prototypes of these applications were used in several fieldwork projects, including a giraffe behavioural study in Kenya, where both their practicality and usefulness in the field were amply demonstrated. More specifically, the giraffe ecologist was able to carry out more work, and new types of work, with less effort and in less time than possible before.

The experiences and feedback generated from using the prototypes in the field radically altered our concept of what a stick-e note is and provoked new ideas in areas such as different triggering strategies, methods to model pretend contexts, and user interface design. The user interface issues in particular were unexpected, yet found to be a critical part of the success of the tools. As a result the next chapter is devoted to exploring the HCI aspects of our work on the fieldwork tools.

Chapter 4:

HCI in the Field

In experimenting with the prototype fieldwork tools described in the previous chapter we found, not surprisingly, that user interface design was a critical element to their successful deployment in the field. Good HCI design considers the abilities of both the user and computer and how they can best be joined through the user interface to effectively carry out a particular task. The majority of HCI research to date has concentrated on the desktop environment where the user is able to focus the majority of their attention on the computer in order to carry out the task at hand. However, in fieldwork environments, and in many other general usage environments for context-aware computing, this assumption no longer holds true. The user now finds themselves in a dynamic environment where the computer will most likely be one of many sources to be demanding their attention. Therefore, the way we design user interfaces for such devices needs to be carefully reconsidered; simply reusing principles developed for desktop computing is unlikely to be successful.

This chapter addresses these issues by exploring the unique characteristics of the fieldwork user and environment, and presents some new user interfaces that were created for the fieldwork tools in order to better suit those characteristics. The chapter concludes with some general HCI design guidelines that we have developed from our practical experiences

4.1   The Very Mobile Nature of Fieldwork

Handheld computing appliances are typically envisioned as tools within the businessperson's domain, where the executive is accompanied by a subset of their business data stored on their handheld computer. During a meeting at the office or whilst commuting to work on the train, the handheld computer allows them to work with their data at a location of their choice. However, the world of the businessperson

is far removed from the environment of the fieldworker. Perhaps one of the most striking differences can be seen in terms of usage patterns. The businessperson will normally be seated at a desk to use their handheld computer, or perhaps with the handheld computer rested on their lap. We could therefore describe this as portable computing rather than truly mobile computing because, although the user can roam anywhere with their handheld computer, it is generally with the intention of bringing computing resources to use within a *static* workplace rather than to use them whilst on the move. The fieldworker's environment, however, is a much more dynamic one, where the handheld computer will be utilised throughout the course of the user's work, often spread over a wide geographic area. That is, the usage of the handheld computer is truly *mobile.*

Static usage of handheld computers pose HCI challenges centred around the problems arising from the ever diminishing size of the hardware, e.g. examining how software displays can be adapted to the dramatically smaller handheld computer screens as in [Kamba, Elson 1996 - #71], [Sarkar, Snibbe 1993 - #122] and [Scholtz, Lockhart 1997 - #129]. However, at least the environment of use is still in common with traditional desktop or laptop computers. *Mobile usage* of handheld computers offer even more challenges as not only do the issues of miniaturisation have to be addressed but also the completely different environments of use too. We believe that the requirements of computing hardware and software intended for mobile usage are significantly different from that of their statically used counterparts, and it is these different requirements and how to satisfy them that we are interested in.

Kristoffersen et al [Kristoffersen and Ljungberg 1999 - #75] suggest three categories of mobility: wandering (where the user wanders around with no one specific destination as part of their day to day work), travelling (where the user spends some time travelling in a vehicle in order to get to a particular location), and visiting (where the user may spend some time in different locations). They also classify the technology used into mobile, portable and desktop categories. However, these classifications focus on the mobility of the *user* and *device*, whereas we are more interested in the mobility of the *activity*. We believe it is the amount of mobility that the user requires whilst simultaneously using a device that is the primary factor in influencing its design: hence our classification of static usage and mobile usage.

The ecology fieldwork work described in the previous chapter has given us a good case study of mobile usage requirements. However, the user interface concepts and prototypes that we propose in this chapter are intended to be widely applicable and are not solely aimed at this area. Indeed, much of our work is valid for applications that require mobile usage but are outside of the fieldwork arena altogether, e.g. handheld computer tourist guides [Cheverst, Davies 2000 - #21], tools for a field service engineer [Kristoffersen and Ljungberg 1999 - #74], etc.

4.2 Four Characteristics of the Fieldworker User

The nature of fieldwork has been described in general terms as highly mobile, where the fieldworker will use the handheld computer throughout a variety of environments during the course of some work: typically data collection activities. The unique nature of mobile usage requirements within this context can be defined as four characteristics:

➢ **Dynamic User Configuration.** The fieldworker will want to collect data whenever and wherever they like, but it is extremely unlikely that there will be any chairs or desks nearby on which to set-up their computing apparatus. Nevertheless, the fieldworker will still want to record data during observations whether they are standing, crawling, or walking (all of which would be quite normal in fieldwork conditions).

➢ **Limited Attention Capacity.** Data collection tasks are oriented around observing a subject. Depending upon the nature of the subject the user will have to pay varying amounts of attention to it. 'Snap-shot' observations require little more than recording the current state of the subject at a particular point in time. However, many observations are carried out over a more prolonged period of time during which the fieldworker must keep constant vigil on the subject to note any changes in state, e.g. observing animal behaviour. In these situations the user needs to spend as much time as possible in observing and to minimise the time devoted to interacting with the recording mechanism.

➢ **High-Speed Interaction.** The subjects of some time-dependent observations are highly animated or, more commonly, have intense periods or 'spurts' of activity. The fieldworker is normally a passive observer whose work is subject-driven;

therefore during these spurts of activity they need to able to enter high volumes of data very quickly and accurately, or it will be lost forever.

> **Context Dependency.** The fieldworker's activities are intimately associated with their context or, if different, the subject's context. For example, in recording an observation of a giraffe, its location or the location of the observation point will almost certainly be recorded too. In this way the data recorded is self-describing of the context from which it was derived. Further applications of the data often involve analysing these context dependencies in some form, e.g. plotting giraffe observations on to a map if location is part of the context.

The relative importance of these four factors can vary with different fieldwork. For example, in testing our prototype software we have been involved with two projects: a giraffe observational study in Kenya as detailed in the previous chapter [Pascoe, Morse 1998 - #99], and an archaeological survey near Sevilla, Spain [Ryan, Pascoe 1997 - #119]. The giraffe behavioural study strongly exhibited all four of the above characteristics, whereas in the archaeological study the characteristics of limited attention capacity and high-speed interaction were not so pronounced. The differences lie in the nature of the data collection subject; giraffe are very animated whereas roman pottery is quite static. However, these attention and speed factors *are* still of importance in archaeological fieldwork because, although the pottery may well be fixed in absolute terms, the archaeologist will walk around an area and note any interesting subjects that he passes by. Therefore, relative to the observer, the focus of observation is changing quite rapidly, and the amount of attention that can paid to observations, and the speed of recording them, are limiting factors as to how quickly the fieldwork can be completed.

As an example of parallel research to ours, the work of telecommunication service engineers and maritime consulting staff is examined in [Kristoffersen and Ljungberg 1999 - #74]. These mobile workers exhibit many of the same characteristics as our fieldworkers; indeed, climbing up a telegraph pole to fix a telecommunications installation is not so different to climbing up an acacia tree to observe a giraffe in terms of the mobility requirements of a handheld computer, except perhaps for the high-speed interaction requirement. However, the authors have chosen to focus on identifying the general characteristics of the work context rather than the general

characteristics of the user as we have done. We believe that the different approaches are complementary and both highlight the limited attention capacity and dynamic configuration of the user, though our work also addresses the typically context-dependent nature of mobile work.

4.3 The Features of a Prototype Fieldwork Tool

We have concentrated on developing novel software applications that utilise existing low-cost hardware rather than employing custom-made or expensive equipment, making it more attractive for our user groups to consider deploying. Hence our use of the PalmPilot and GPS receiver as described in the previous chapter. However, although we were not able to design the hardware ourselves, we carefully examined the available devices and evaluated their suitability for fieldwork environments through the following criteria:

➢ **Pen user interface.** We found that the flip-open 'clam-shell' pocket computers equipped with miniature keyboards were not suitable for fieldwork environments, where the user is typically standing or walking whilst operating the device. Although ideal for static situations where they can be rested on a work-surface, in-hand use of these devices requires both the user's hands and often involves a clumsy method of typing with one's thumbs. Pen-based interfaces on a pad-like device provide a more ergonomic solution that can be held in one hand if simply viewing data; they generally use some form of handwriting recognition for entering data. They provide a natural substitute for the fieldworker's paper notebook: similar in size and operation, and suitable for use by the user in many different dynamic situations (e.g. whilst walking).

➢ **Small and unencumbering form factor.** Given the nature of some of the work it is important that the equipment does not encumber the user's body or senses in any way. For example, in searching for an elephant in dense bush the user will use their sense of sight and hearing to the full and, should the elephant give chase, will need to be able to run to the best of their ability! Therefore we have limited our equipment selection to small handheld computers, ideally of a size that would unobtrusively fit into a trouser pocket when not in use.

➢ **Battery-life.** A typical fieldworker will spend a day in the field before returning to a base camp. Therefore, a device that can be used for at least a whole day without requiring replacement batteries is desirable.

➢ **Robustness.** The very nature of the environment makes it necessary to have devices that are able to cope with knocks, drops, and the general conditions of outdoor life, including heat, dust, rain, etc. In short, a very durable device is required.

➢ **Connectivity.** Sensors may be used to automatically obtain some data values if there is a way of connecting them to the main device. However, the process of data collection is not an end in itself. The collected data will need to be downloaded to a desktop computer for analysis and detailed study once the fieldwork has been completed. Therefore, it is necessary to have a device that can be easily connected to both a PC and various sensor devices.

Based on these criteria we chose the 3Com PalmPilot as the most suitable device at the time. There are a number of specialised manufactures of ruggedised mobile computers, but we wished to select a device that was reasonably priced, widely available, and suitable for a variety of mobile environments, not just in fieldwork. All of our software tools for ecology fieldwork (as described in the previous chapter) have been developed for this platform. The simple form-based interface of our prototype software embodied the design philosophy of PalmPilot software: "if it needs a manual then it's too difficult to use". Rather than providing a radically new interface design from what the ecologist may have previously encountered, we instead sought to provide innovative features set within a familiar interface metaphor. This approach allowed the users to quickly master the use of the system.

We are aware of one other project that aims to utilise computer-based fieldwork tools in African ecology work [Bailey 1997 - #6]. However, their interests lie in allowing illiterate trackers to accurately record the location of animal sightings through a simple pictogram interface, rather than assisting the ecologists to record large and complex data sets. So the user interface issues are quite different. They are interested in providing trackers with a recording tool where they can note down any animal sightings (and possibly their behaviour at the time of being spotted) in as quick and easy a

manner as possible using a visual pictogram-based interface. Our work instead focuses on providing ecologists with tools that aid them in recording much more complex observations, such as giraffe behaviour time series, with the emphasis on developing user-interfaces that are as transparent as possible to the observational activities.

We tested our prototype system in a number of environments, the most rigorous of which was the three-month behavioural study of giraffe in Kenya [Pascoe, Ryan 1998 - #103] [Pascoe, Ryan 1998 - #101]. In this trial a willing ecologist, Kathy Pinkney, replaced her paper notebook with our prototype for the entire period of her fieldwork, using it for *all* of her data collection tasks. The focus of her research was to investigate the feeding behaviour of giraffe in order to assess their impact on the vegetation within the Sweetwaters game reserve. To do this effectively she needed to collect a large amount of raw observational data of giraffe feeding. For each nibble of the branch that a giraffe took, Kathy would have to record the location, time, tree species, feeding height, and number of bites taken, in addition to any other activities the giraffe indulged in in-between nibbles. The giraffe's voracious appetite for acacia and general restlessness make recording the observations very difficult, especially considering that the ecologist needs to look through a telescope and operate a stop watch at the same time. In fact, such tasks are so difficult in traditional paper-notebook recording methods that an assistant note-taker is normally required. However, with the aid of our prototype fieldwork tools Kathy was able to collect all the necessary data single-handedly.

During the course of the three-month study the prototype performed at a level that allowed the ecologist to complete more work, in a way that was both faster and easier, than is possible in a manual system. At the end of the study approximately 6000 observations had been recorded. The HCI factors in the prototype that led to this success can be formulated into two general principles:

➢ **Minimal Attention User Interfaces.** Providing interface mechanisms that minimise the amount of user-attention, though not necessarily the amount of user-interaction, that is required to perform a particular task.

➢ **Context-Awareness** [Schilit, Adams 1994 - #125]**.** Imbuing the device with the capability to sense its environment.

The remainder of this chapter describes how these principles were applied in the prototype system and presents our work on further enhancements and research arising from our experiences of employing these principles in the field. We first examine context-awareness, but only the small subset of the concept that is directly related to the HCI issues of the fieldwork tools and users.

4.4   Context-Awareness

The fieldworker is generally equipped with a plethora of equipment to assist in the observation process. In the Kenyan study, for example, a map and compass would have been required to pinpoint location and a stopwatch required for recording time series data such as giraffe behavioural observations. However, rather than taking more equipment out into the field, we believe that a fieldworker endowed with a handheld computer will actually take out less. The reason for this apparent paradox is that we wish to assimilate as many of the other equipment interfaces into the handheld computer as possible and to automate their operation. In addition, instead of providing an electronic 'copy' of the device (e.g. a graphical simulation of a GPS receiver on the handheld computer's screen), we aim to embed the appropriate function within the task it is related to: for example, automatically entering the current location into a new observation note. This is achieved by making the handheld computer aware of its context through various attached or embedded sensors so that it is able to supply context information when needed.

In our prototype we made our programs aware of two elements of their context that are useful in a wide array of activities: time and location. Knowledge of time is easily obtained through the unit's own internal clock, and this can be used to provide various timing functions that eliminate the need for a stopwatch. Location was provided through an attached GPS receiver that could pinpoint the user to within 100 metres anywhere in the world (given an unobstructed view of the sky). The GPS receiver was a separate unit attached via a serial cable, but we expect problems with cabling to disappear in the near future through a combination of more sensors embedded in the devices themselves and the ability to communicate with other external sensor units through wireless technologies such as Bluetooth [Bluetooth-Consortium 2001 - #11].

In effect, the handheld computer could become a universal interface or remote control for a number of sensor 'black-boxes' that are wirelessly connected.

From a few basic sensors a number of software-derived context elements can be generated. For example, a tide-level context element can be computed from the location and time context elements. Similarly, a 'dominant vegetation' context element can be derived from a location and GIS vegetation map. In fact, the wealth of potential context information has spurred us into developing a Context Information Service (CIS) [Pascoe, Ryan 1999 - #102], which is presented in the chapter 6.

One of the characteristics of the fieldworker described earlier was the need for high-speed interaction. Context-awareness can help in this area by automating some aspects of the fieldworker's activities. In the prototype software the StickePad automatically defaulted any time or location fields of a newly created note to the current clock or GPS reading respectively (see Figure 23). Even such a seemingly minor enhancement made the ecologist's job much easier. For instance, recording giraffe feeding behaviour through a telescope would normally have required two people, one to dictate the observations being made through the telescope and the other to use the stopwatch to record the times and details of the rapidly occurring events. However, the prototype system allowed a single person to perform both tasks by automatically completing the timing information as soon as the user indicated a new event had taken place, leaving them to simply enter a code for the behaviour.
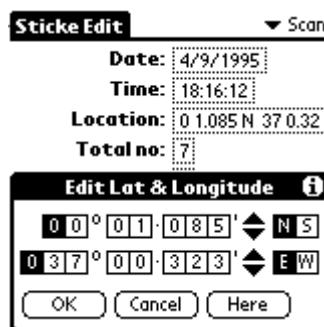
Figure 23. Editing a location field in the StickePad illustrates how context-awareness can be used to expedite data collection; in this case by automatically entering the user's location (derived from an attached GPS receiver) and allowing it to be easily updated via the 'Here' button.

Another characteristic attributed to the fieldworker is their context dependency. As mentioned earlier, the data being collected is effectively a description of various

elements of their context and at a later date the complete collection of data will be compared and analysed from the perspective of one or more of these context elements. For example, the collected notes could be plotted in a GIS in order to visualise and analyse the data from a location context element perspective. Equipped with a handheld computer we can effectively bring cut-down versions of these context visualisation tools into the field, where not only can we view the various notes that have been recorded but also our presence relative to them. In the prototype we implemented a StickeMap program to demonstrate a form of context visualisation by plotting the recorded notes and user's current position on to a configurable map. Visualising data using context information provides a powerful mechanism that allows the user to gain an overview of the data from a particular contextual perspective, to filter information that they are interested in, and also to look for patterns in the data.

4.5   The Minimal Attention User Interface (MAUI)

MAUIs seek to satisfy the needs of the fieldworker with respect to their characteristics of dynamic user configuration and low attention capacity. An example of a task in the Kenyan fieldwork that illustrates both of these characteristics particularly well is the detailed giraffe observation. During one of these observations the ecologist was often hiding behind vegetation, walking through the bush, or crouching over a telescope. Data needed to be recorded in any of these circumstances. Additionally, observing a giraffe's detailed feeding behaviour (such as the number of bites taken from a particular acacia tree) required a great deal of attention. This is especially true when observing from a distance through a telescope, where, unless the user pays constant attention, the giraffe can quickly move out of the field of view.

Conventionally, handheld computers require the direct attention of the user for the whole duration of a particular task. During this period all of the user's attention is focused onto the device. For example, to select a document the user will hold their handheld computer in one hand, select the document with the pen held in the other, and all the time be looking at the device in order to correctly operate the interface. In a fieldwork environment this distracting process can negatively affect the quality of the work. Note that it is not the number of interactions occurring that is the important factor, but the amount of attention that they require from the user.

A MAUI attempts to remedy this situation by transferring interaction tasks to interaction modes that take less of the user's attention away from their current activity. In effect it is about shifting the human-computer interaction to unused channels or senses, and in a way that is not so cognitively demanding to distract the user from the task at hand. As a small experiment of this idea, our prototype fieldwork software overloaded two of the hardware buttons of the PalmPilot device with a configurable increment and decrement function. These buttons could then be used to manipulate sequential data with less attention from the user because the buttons provided enough tactile feedback without requiring the user to actually look at the device. The user could configure the amount decremented or incremented by these buttons for particular types of data (e.g. tree height may increment in units of five metres and giraffe bites in steps of one). This feature was most usefully employed in counting giraffe bites from a tree: here the ecologist could keep a running total of the number of bites taken - just by clicking a button for each bite - whilst simultaneously observing the giraffe through the telescope. In effect, this is an *eyes-free* form of human-computer interaction.

Figure 24. The ecologist observes the giraffe whilst simultaneously recording data on the PalmPilot using the Minimal Attention User Interface.

We are exploring other methods to optimise the eyes-free MAUI of our prototype. The touch-sensitive screen provides one opportunity. If divided into selectable areas, say four quadrants, a particular function or data value can be assigned to each of the quadrants, e.g. a tree species selector where top-left = acacia, top-right = uclea, bottom-

left = scutia, bottom-right = other. The user can easily identify the four corners of the screen with their thumb and hence operate the interface in eyes-free mode (especially if some form of audio feedback is given). Although we may not be able to divide the screen into enough areas to support all functions or data types we can certainly implement the most frequently required options to optimise for an eyes-free mode of operation. However, lack of tactile feedback from the touch screen may be of concern.

Our interest in MAUIs is not limited to eyes-free forms of interaction but also covers other methods that attempt to minimise the amount of distraction caused to the user's activity. *One-handed operation* is such a method. Although the user may need to occasionally look at the screen, one-handed operation allows them to operate the device with one hand whilst continuing to perform tasks with the other. Such a facility is useful in many diverse situations, not just in fieldwork. For example, consider the businessperson who wishes to consult their diary and to-do lists for the day whilst walking to work with their briefcase in one hand and their handheld computer, retrieved from their pocket, in the other. In such circumstances the hand that holds the device also has to perform the interaction tasks. Small devices such as the PalmPilot are ideally suited to this type of activity as they can be easily held in one hand whilst leaving the thumb free for manipulating the screen or buttons.

Thus far we have only discussed operating on individual portions of a complete data set or task. We also inherently need a minimal attention method of navigating between these individual features. This was the area of the prototype interface that proved least successful in the first Kenyan trial. For example, to edit a note the user had to decide which field of the note to select, perform the editing operation using the type-specific controls, return to the list of fields, and then decide the next field to edit. Although easily executed, a problem arises when the user's concentration is directed elsewhere, such as at a giraffe: the number of decisions and manipulative processes involved in selecting and modifying the data appropriately become a distraction to the main task at hand. To improve this situation the form-filling interface could be structured as a set of 'layered' screens, one for each data element. The user would then be presented with a sequence of these screens that is optimised for minimal attention, much like filling in a questionnaire question by question. Note that, as with many of the features presented, this would be an optional enhancement to the existing system rather than a

replacement, because if the user's attention is not pressed then they may prefer to work with the data in the 'bigger picture', e.g. viewing the whole form whilst editing a small field within it.

We have attempted to combine our ideas of layered sequential screens with those of eyes-free or one-handed controls in the prototype fieldwork tools in order to provide a better MAUI (i.e. one that can be operated with even less distraction to the user's activity). This has been tested in another field trial with the ecologists in Kenya. This time the fieldwork requirement that drove our designs was the need for rapid data collection carried out in parallel with other manual activities. An example that illustrates this need is elephant dung pile counting. In this task the ecologist briskly walks a ten kilometre transect recording the location and number of elephant dung piles on each 500 metre leg of the journey. They may also record any other animal sightings of interest along the way and will often be using their binoculars. We concentrated on providing a one-handed mode of operation for such activities so that the fieldworker could easily operate the prototype whilst walking, looking through binoculars, tracking animals, etc.
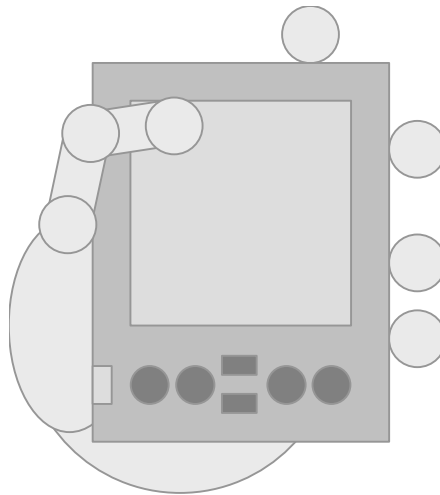
Figure 25. Most comfortable grip for one-handed operation.

The first design decision in developing our one-handed MAUI for the fieldwork tools was to find the most comfortable fit in the hand for the device and to select the control surfaces we would use. Figure 25 shows the most comfortable and secure method we found to hold the device with one hand, which leaves the thumb free to operate the device. However, the touch screen and hardware buttons can not be used together as

the hand can only be positioned to place the thumb exclusively on either control surface: the thumb is not extensible enough to be able to use both surfaces from the same hand position. We decided to use the hand position that favours the touch screen as this control surface offers much more flexibility in the design and presentation of the MAUI controls.

In such a hand grip the thumb is more easily manoeuvred in the vertical plane than the horizontal, so the one-handed MAUI constructs its control elements as a series of stacked lateral bands, equally suited to both left and right-handed users. The main screen of the MAUI-enabled StickePad, as shown in Figure 26, consists of three buttons that can be activated with a tap of the thumb to select the appropriate function.

StickePad
Main Menu:

New Note
Edit Note
Delete Note
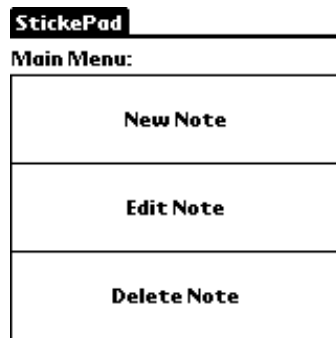
Figure 26. StickePad main screen in one-handed mode.

Selecting the 'New Note' button invokes a sequence of one-handed controls through which the user progresses to create a new note, as shown in Figure 27. In the first control (a list selection control) the user chooses a note template by scrolling through the possible choices using the up and down arrow buttons, then pressing the central button when it displays the desired template name. The remaining controls in the sequence allow the user to enter data as appropriate for the note. The example used in Figure 27 shows a time control, which that can be updated with the current time, and a numeric control that can be incremented and decremented using the arrow buttons. As with the list selection control, once the desired data value is shown it is tapped with the thumb to select it. Controls exist to manipulate all the Stickepad's data types except for textual data. A control does exist to display the textual data but it uses a normal text editing dialog box if it needs to be edited, as editing text with this one-handed method is probably too cumbersome to be useful.
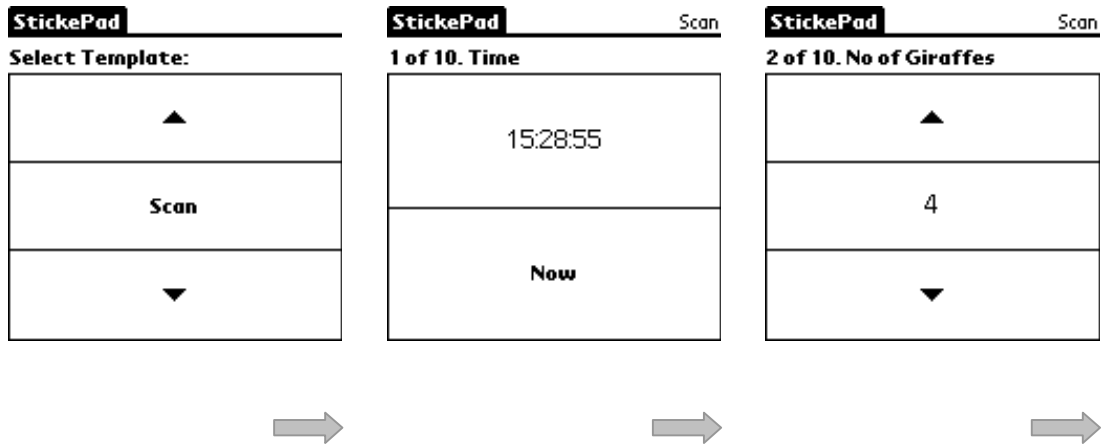
Figure 27. The first three of a sequence of one-handed controls to enter a new note.

In addition to designing an optimal layout of controls for one-handed operation we have also investigated the use of multiple taps. Using this technique a button may have more than one function assigned to it depending on the number of taps of the user's thumb on the touch screen. This is equivalent to the single and double-clicks commonly executed on mouse buttons. A single and double tap can be easily performed on the touch screen, and even treble and quadruple taps. However, both the physical operation and cognitive load of remembering the number of taps for different functions make it prohibitive to exceed quadruple taps. In the prototype MAUI we commonly make use of single and double taps, and use treble and quadruple taps more sparingly for functions that are used less often. One of the most useful examples of using the multiple taps in the StickePad is to navigate through the sequence of controls used to edit or enter each field of a note, as shown in the examples given in Table 15.

The multiple taps operate on the particular part of the control to which they are applied, e.g. double clicking on the numeric control's up arrow for incrementing is obviously different to clicking on the numeric control's down arrow for decrementing. To navigate through a sequence of controls, multiple taps are directed at the data display part of a control, e.g. the number displayed in the numeric control.

There is one other one-handed input technique: the thumb stroke. This involves running the thumb from the top of the screen down past the halfway mark and also the reverse action starting from the bottom of the screen. This is a very fast and fluid motion and is used in the StickePad program to quickly flip between one-handed mode

and normal mode (where the user may be interested in looking at the overview of collected data displayed on the main screen). The thumb stroke is available to the user whatever the current mode. Importantly, this allows the user to effortlessly switch to one-handed mode using a one-handed operation rather than needing to activate the menu and then select the option from a menu item in normal mode (which would require two hands).

Number of Taps	Sequence Navigation	Number Editing
1	Next field/control in sequence.	Increment or decrement (depending on which button pressed – up or down arrow) by 1.
2	Previous field/control in sequence.	Increment or decrement by the number-specific step amount.
3	Jump to the first field/control in the sequence.	-
4	Jump to the last field/control in the sequence.	-

Table 15. Two example uses of multiple taps.

In general, we are designing interfaces based on a model of the fieldwork activity, not on a model of the fieldwork data. Although some interface elements are created to manipulate particular types of data, it is the user's activity that shapes the design of our interface. Identical data may have completely different interfaces depending on the user's activity. For example, in the field the ecologist may want a simple and direct interface that is oriented to recording data quickly, whereas back in the laboratory the user will likely view the same data in a much more rich and complex form, such as in a GIS (Geographical Information System). It may be tempting to encode the visual interface into the logical template descriptions (as is often the case with HTML documents) but this severely limits the flexibility and portability of the data model. Therefore, we have been careful to keep interface and data models distinct in our prototypes.

One final opportunity in developing MAUIs for fieldwork lies in the new hardware capabilities that are becoming available in-built or as attachments to handheld

computers. For example, microphones are becoming more prevalent with handheld computers and provide opportunities for voice recognition, whilst small vibrating units (typically used by pager software) offer the developer another means of providing feedback to the user. Other innovative forms of hardware interface mechanisms are explored in [Harrison, Fishkin 1998 - #57] and [Rekimoto 1996 - #109], which investigate how various tilt and touch sensors can be employed to manipulate the device through a variety physical movements and gestures.

4.5.1 General Guidelines for the Selection or Design of MAUIs

The design of our MAUIs is strongly influenced by the characteristics of the ecology fieldwork activities. In particular, the interaction modes that the MAUI utilises are selected to best fit the characteristics of the fieldwork activities. In order to assist in a more general understanding and comparison of how different activities affect the choice of interaction modes in a MAUI, we can construct a table of the activity's interaction mode characteristics, as the example in Table 16 illustrates.

	Visual	Audio	Tactile
Environment **(African Bush)**		*Restricted.* Cannot make much noise in case animals are disturbed.	
User **(Ecologist)**			
Task **(Giraffe observation)**	*Prohibited.* Ecologist will need to keep her eyes on the giraffe under observation.		
Tools **(PalmPilot and data collection software)**	Small screen (160 x160 pixel).	No microphone. Very limited audio output through internal speaker.	4 tactile hardware push buttons. Touch screen.

Table 16. Interaction mode characteristics for the ecology fieldwork activity.

In order to more precisely describe exactly where in the activity the interaction mode characteristics originate, we have divided activity into environment, user, task, and tools. Each of these aspects of the activity may prefer, restrict, or prohibit the use of a particular interaction mode.

In the ecology fieldwork activity in Table 16 we can see that the use of the audio mode in a MAUI would not be a particularly good choice given the restriction imposed by the environment and the very limited audio capabilities of the tools. Additionally, the giraffe observation task requires the full attention of the user's eyes, prohibiting the use of the visual mode in a MAUI. However, there are no restrictions imposed on the tactile mode by any aspect of the activity: in fact the tools even offer a few methods of exploiting this mode. Therefore, in selecting or designing a MAUI for this activity we would seek to choose one that primarily exploited the tactile mode.

Comparing the interaction mode characteristics of our MAUIs (Table 17) with that of our activity (Table 16) enables us to select the eyes-free push-button MAUI as the best interface for the ecology fieldwork activity. However, we also need to make sure that the data types that need to be entered and conveyed in the activity are supported in the chosen MAUI (in the fieldwork example, the majority are).

		Visual	Audio	Tactile	Data Types
Eyes-Free Push-Buttons	**Input**			Hardware buttons.	Numeric.
	Output				
One-Handed Oversized Controls	**Input**			One-handed thumb-driven input via touch screen.	Numeric, picklist, location, time, date, bearing, data series.
	Output	Easy-to-read oversized controls.			Numeric, picklist, location, time, date, bearing, data series.

Table 17. Interaction mode characteristics of our MAUIs.

Complex activities may span more than one environment, user, task, or tool. If these present incompatibilities in interaction mode characteristics then the activity needs to be divided into sub-activities in which the interaction mode characteristics are compatible. An appropriate MAUI can then be designed or selected for each sub-

activity. Additionally, if the MAUI designer is able to choose the tools that are used in the activity then they will be selected to suit the chosen MAUI rather than the MAUI chosen to suit the tools. The tools only appear on the activity's interaction mode characteristic table if they are a fixed part of the activity, as the PalmPilot effectively was in our case due to our development investment in it. If we could have chosen any tools then perhaps a device with a chording keyboard would have been our selection, since it allows a more sophisticated use of the tactile mode.

In summary, the 'minimal attention' in the 'minimal attention user interface' is achieved through the effective use of modes of interaction that least distract or interfere with those that the user is employing in her current activity. This complementary use of interaction modes is the key to successful MAUI design.

MAUIs tend to be constructed for specific activities rather for general-purpose use, in both our work and others: for example, in developing navigation aids for the blind [Strothotte, Fritz 1996 - #137] and assistants to field service engineers [Kristoffersen and Ljungberg 1999 - #74]. By constructing tables of interaction mode characteristics for these activities we are able to more rigorously compare them with other activities, and to select or design an appropriate MAUI that uses complementary interaction modes. For example, considering the navigation aid to the blind, the following table can be constructed.

	Visual	Audio	Tactile
Environment (Urban area)			
User (Blind person)	*Prohibited.* No visual sense.	*Restricted.* Don't want to interfere with the person's awareness of their environment.	
Task (Navigation)			*Restricted.* Person may be carrying shopping, a cane, etc.

Table 18. Interaction mode characteristics for navigating for the blind.

Note that the table includes no tools aspect to this activity because we have assumed that we have a choice of technology. On comparison with the table for the ecology

fieldwork activity we can see that the prohibition of the visual mode is shared. However, in this activity the audio mode is less restricted than the tactile one, which could be prohibited altogether in certain situations (e.g. when carrying a lot of shopping). Therefore, a MAUI that primarily utilised the audio mode, with perhaps some auxiliary use of the tactile mode, would be best suited for a navigation aid for the blind. Similarly, Table 19 shows that the activity characteristics of the telecommunications service engineer also makes an audio-based MAUI the best selection, though for different reasons.

	Visual	Audio	Tactile
Environment **(Up a telegraph pole)**			*Restricted.* User needs at least one hand free to hold on to the pole.
User **(Field engineer)**			
Task **(Field servicing)**	*Restricted.* The user will wish to keep her eyes on the facility under repair for most of the time.		*Restricted.* May be carrying lots of other equipment.

Table 19. Interaction mode characteristics for a telecommunications field servicing.

We hope that by explicitly identifying interaction mode preferences/restrictions and their origins in this way will enable developers to more easily and accurately identify similar activities in terms of their interaction mode usage (the applications could appear quite different in other ways). It should also assist in quickly identifying existing MAUIs that can be successfully deployed for a particular activity, or in specifying the interaction mode requirements or limitations for the design of a new MAUI.

4.6   Summary

We believe that context-aware tools will generally be exploited in highly mobile and dynamic activities. It is these very different activities that drive the development of radically different types of computing devices (e.g. handheld or wearable computers) and the need for different modes of interaction with these devices. More specifically, in this chapter the mobile nature of fieldwork activities was examined and four general

characteristics of the fieldworker identified. These characteristics illustrated how HCI design for desktop-computing environments is ill suited to computing in the field.

Two new general HCI principles for mobile environments were presented: context-awareness and minimal attention user interfaces (MAUIs), both derived from the successfully deployed fieldwork tools described in the previous chapter. In this chapter we focused primarily on an exploration of MAUIs, describing their aim to reduce user attention but not necessarily user interaction. Two different types of MAUI were developed: one for one-handed interaction and another for eyes-free interaction; each is suited to different types of activity. In conclusion some general guidelines were developed in order to assist user interface designers in the selection and development of MAUIs for a given activity.

Chapter 5:
Fieldwork Tools Usability Study

Many existing context-aware applications have never seen real use over an extended period. However, we thought it important, in order to establish the validity of our ideas, to conduct proper field trials. Although we had already trialed our various prototypes in the field, until the summer of 1999 they had only been used by one or two ecologists at a time. What is more, these ecologists had already expressed their interest in using such tools before the trial began. Therefore, our test user group was somewhat predisposed to liking the tools and were tolerant of many problems that no doubt a less enthusiastic user might have found unacceptable. In order to more accurately gauge the usability and usefulness of the fieldwork tools we decided to perform a more extensive trial with a much larger and unbiased user group. We wanted this group to be ambivalent towards using the equipment before the trial, and to consist of a cross section of people ranging in ability from computer professionals to those that had barely even touched a computer before, i.e. providing a representative sample of potential fieldwork users. Our hope was that the fieldwork tools could be mastered and usefully employed by any of the individuals in the test group, and that users would express a preference for our prototype fieldwork tools over a conventional paper-based system.

5.1   The Ecology Project and Test Group

Our need to perform a more extensive usability trial happily coincided with the start of an extensive ecology project at the same Kenyan game reserve in which the earlier single-user field trials had taken place. The aim of the ecology project was to construct an ecological model of the whole reserve, with the key task of determining the ecological carrying capacity of black rhino within the reserve. To construct such a model, large amounts of data need to be collected over many years, and the only way in which this data can be collected is by sending trained teams into the bush to take many measurements and observations at regular intervals.

The ecology project was funded by a charitable organisation called Earthwatch, which finances hundreds of fieldwork projects all over the world. They support projects through revenue generated by volunteers, who pay to take part in the fieldwork projects for a two-week 'holiday'. Thus Earthwatch provide both the money and fieldwork man-power to support a project. Projects are run by a small team of professionals in the field.

In the case of the Kenyan Earthwatch project, called "Kenya's Black Rhinos", five teams of approximately ten volunteers came out over the course of the summer of 1999 to perform the fieldwork tasks for the two-strong ecology team (consisting of the principle investigator, Alan Birkett, and his assistant, Kathy Pinkney). Each team stayed at the reserve for two weeks and performed various fieldwork activities over that time.

The fifty volunteers over the course of the summer formed our usability test group. Coming from all walks of life, different age groups, different countries, and with a wide range of computing abilities (their interests lying in ecology rather than computers), they made an ideal unbiased and diverse group of people on which to perform a usability test. To gauge their opinions of the prototype we designed a questionnaire to capture their previous experiences of fieldwork and/or computing, and their experiences and reactions to using the prototype in the field. By analysing the results of this questionnaire we would assess the success, in terms of usability, of the fieldwork tools.

5.2   The Fieldwork Activities

In order that the trial results be seen in perspective, we now explain how the fieldwork was conducted. Each team of ten volunteers were split up into five pairs for the duration of their two-week stay. With one day for arrival and another for training, this left twelve days for the actual fieldwork itself. On the training day the group of volunteers was given an introduction to the project, the reserve's eco-system, and training on the various devices they would use and the activities they would be expected to perform. Of the full day's training, only half an hour was explicitly devoted to explaining the operation of the prototype fieldwork tools, i.e. operation of PalmPilot, GPS receiver, and software. However, the volunteers were given the opportunity to use the tools in a practice tree measuring activity later in the day. After the full day's

training the volunteers were expected to be sufficiently proficient to carry out any of the different activities without assistance, including use of all the equipment.

Eight different activities were devised for the volunteers, each of a day's duration. All of these activities were designed to produce data for the overall ecological research programme, and all of them utilised the prototype fieldwork tools for data collection. A rotating schedule was drawn up so that the pairs did different activities on each day; though some of the activities were repeated. Appendix I describes each activity and how the prototype was used to support them.

5.3   Advantages of Using the Prototype for the Project Organisers

Our main concern with the usability study was to ascertain how the prototype fieldwork tools benefited or hindered the volunteer's fieldworker activities. However, their use also has a very significant impact on the role and work of the managers of the ecology project. Indeed we were aware of this from the start of the project and it was the many positive benefits that would be derived from using the technology that motivated the project principal investigator's desire to use the equipment.

If the project had been run without the prototype fieldwork tools then each volunteer would have been equipped with a pen and paper notepad and would have been asked to record the data manually. The "disaster stories" of this method were what spurred the project's principal investigator into using our tools in what was the first field season of their fieldwork. The benefits of our prototype system that the principal investigator saw are listed below:

➢ **Automatic Transcription.** Volunteers were able to download the data straight into the laptop computer and onto the project spreadsheet in a matter of minutes. In a traditional paper-based system someone (either the volunteer or project organiser) would have had to key in all the hand-written data collected, a process that is both time consuming and fraught with errors (especially with some individual's hand-writing). The sheer volume of data collected in this project made the transcription process especially foreboding. The first three teams managed to collect 80,000 units of data during the first six weeks (a spreadsheet cell being a unit of data). Indeed,

the data collection strategy would almost certainly have been altered to reduce the volume of data had a manual system have been used.

➢ **Reliable Measurements of Time and Place.** The location, time and date fields were automatically filled in by the software and hence immune from the introduction of most types of error. The reliability of time and place data is critical as all the other data elements of an observation are often correlated to either one of these fields. Automatic availability also allowed the quick and effortless recording of location data where it probably would not otherwise have been recorded (e.g. to check where the volunteer had been during the course of a day's activity). In a manual system the activities based around behavioural observation or transect walking would have involved much more work from the volunteer and the data collected would be much more susceptible to error.

➢ **Data Consistency.** The ability to define a fixed template of fields to collect a particular set of data, and the facility to define the particular type, parameters, and validity checks for each field, ensured that the data was collected in a consistent and valid form from volunteer to volunteer. With a paper notebook it would have been difficult to enforce any particular data collection regime (as was found on one occasion when one of the PalmPilot devices failed and a paper notebook had to be used instead).

➢ **Data Security.** Downloading the PalmPilot data to the laptop each day allowed an effective backup strategy in which many copies of each day's data were kept in distributed locations (including back in England via email). In the most likely worst case equipment failure scenario only one day's worth of data would be lost.

The deployment of the prototype fieldwork tool does have some disadvantages. Firstly, there is the cost of the palmtop computers (which could only be expected to survive for two years in such harsh conditions). Secondly, the project managers must spend some time training the volunteers in the use of the equipment and providing support. However, only one half hour of training was required during a full day of tuition on various other equipment use and fieldwork activities, so this was not considered much of a disincentive. Similarly, the support of the equipment required only a small amount of the manger's time in logistical activities, the most strenuous of which was probably

checking and recharging the equipment's batteries. In the current state of development there was also the risk of failure in the prototype system. However, I was on hand to fix any problems, and the managers could also revert to the paper notebook system in the case of a catastrophic failure. Thankfully no such failures occurred, and would not occur with production quality software rather than a prototype.

5.4   The Usability Study

The advantages of the fieldwork tools that we have mentioned thus far are mostly functional ones that enable an enhancement of the fieldworker's data collection abilities in terms of the quantity and quality of data collected. It is the project managers that accrue most benefits from these improvements. However, we are just as interested, if not more so, in how the prototype fieldwork tools positively or negatively affect the experience of the volunteers carrying out the work in the field. To this end we designed a questionnaire with the aim of capturing their experiences and feelings about using the equipment in the field, and also to ascertain a little background information about the individual to see if previous computing and/or fieldwork experience made individuals more predisposed to like or dislike the tools. A blank questionnaire sample is included in appendix C.

The questionnaire was given to the every volunteer on the second to last day of their two-week stay. The first team was not surveyed as we used that initial two-week period to test out and improve the processes and equipment use. However, all of the subsequent four Earthwatch teams were surveyed, resulting in the completion of thirty eight questionnaires. These questionnaires were not anonymous: volunteers were asked to write their names on them so that further feedback could be sought on any comments or suggestions that were either particularly interesting or hard to understand. However, we instructed the volunteers to be ruthless in their criticism so that we would have an accurate impression of their likes and dislikes of the prototype tools, which we could then use to help improve the next generation of prototype tools.

When we processed the questionnaires, we generated four types of data:

➢ **Multiple-Choice Responses.** Multiple-choice questions could be entered straight into a spreadsheet with a row for each subject and a column for each multiple-choice question.

➢ **Codified Responses.** Some questions with textual responses were codified to generate, in effect, a 1-5 rating of a particular quality or attribute. In these cases more than one question could also be combined to generate the codified response. For example, the four questions used to gauge previous fieldwork experience and the three questions used to gauge previous computing experience were combined to generate a 1 (expert) to 4 (novice) rating for fieldwork ability and computing ability respectively. We generated those ratings ourselves rather than giving a multiple-choice option to the volunteer so that the ratings would be more standard and less subjective, and also so that we would have more detailed data on their abilities to refer to if necessary. The resulting codified results could be entered into the same spreadsheet as the multiple-choice questions.

➢ **Categorised Responses.** Some questions elicited responses that we could not possibly predict in advance, e.g. likes or dislikes about the prototype. From all the combined responses for a particular question we attempted to create a set of categories of response. Each question was then represented in its own spreadsheet section with a different response category on each row and a different subject in each column. Each cell of the sheet contains a Boolean value to indicate whether the subject expressed a certain category of response. Note that the subject may express more than one category of response, i.e. the categories are not necessarily mutually exclusive.

➢ **Narrative Responses.** Some data of a more narrative nature was left in its original textual form, e.g. suggestions for future improvements to the prototype.

The multiple-choice and codified response spreadsheet is included in appendix D and the codification scheme included in appendix E. The categorised response spreadsheet is included in appendix F and the narrative responses included in appendix G. The rest of this chapter utilises this raw data to answer some usability questions.

5.4.1 User Analysis

The test group was composed of individuals from professions as diverse as court reporters, airport managers, students, personal assistants, attorneys, and so on, with a fairly even spread of ages from early twenties to late fifties (though slightly more on the twenties side than the fifties). The majority were of American origin (65%), roughly a fifth were British (18%), and the remaining fifth were peoples of various nationalities (German, Japanese, Australian, and Swiss). The group was predominantly female (70%).
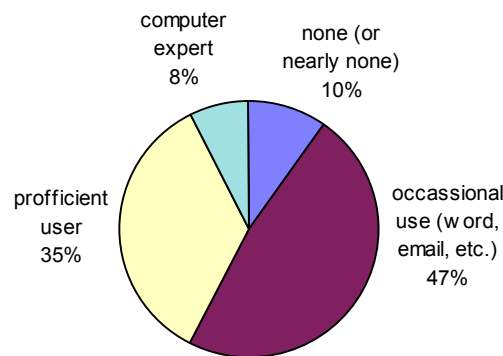
Figure 28. Previous computing experience of volunteers.

In addition to these personal attributes we also gauged previous experience in both computing and ecology through a series of seven questions. For computing experience we categorised individuals as having little or no experience, to be occasional users (if they have occasionally used a computer for simple activities such as word processing or email), to be proficient (if they are experienced users who use a computer on a regular basis), or to be expert (if they have an advanced knowledge of computers and how they work in addition to being able to use them). Approximately half the group were classed as occasional users, one third were proficient users, and the remaining individuals were equally divided between expert users and those with little or no computing experience, as illustrated in Figure 28. Only five of the volunteers had previously used a handheld computer; four of these were PalmPilot users and one was a Psion Series 5 user.

For fieldwork experience we classified the volunteers as having done little or no previous fieldwork, having done one to two fieldwork projects previously, having done many previous fieldwork projects, or to be a full-time fieldwork professional. A third of the volunteers had no previous fieldwork experience, half had been on one or two fieldwork projects previously, and the remaining individuals were equally divided

between many previous projects and full-time fieldwork professional, as illustrated in Figure 29.
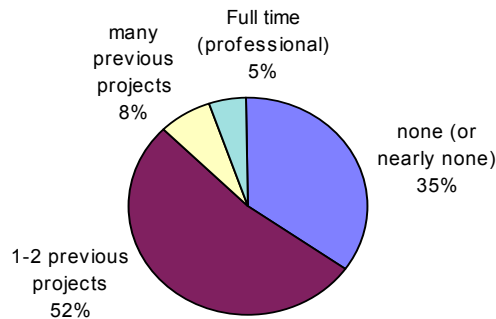
Figure 29. Previous fieldwork experience of volunteers.

As we had hoped, the group was very diverse in nature, giving us a broad cross-section of potential users with whom we could study the acceptance and usability of our prototype fieldwork tools.

5.4.2   Acceptance Analysis

In order to establish how well received the tools were by the volunteers we asked them a number of questions on their acceptability. When asked if they would prefer a manual paper and pencil based system or our prototype system the group was unanimous in selecting the fieldwork prototype. Not a single person chose to use pencil and paper despite the wide range of computing abilities: a certain indicator that the prototype was well received. The were many reasons for preferring the prototype, the top four were data entry being faster (suggested by 22% of the group), increased efficiency with less work and writing (20%), transcription of data being easier (15%), and less chance of error (12%). A complete list of reasons is given in Figure 30.

In learning to use the prototype the half hour of training proved to be just the right amount for almost three quarters of the group, with a quarter stating they would have liked a little more training, but no one requiring a lot more training. The majority of the group felt comfortable using the equipment after the first day (77%) and the remainder within two to three days (18%) bar two individuals. One of these two individuals took a week to get comfortable with the equipment and the other never gained a satisfactory level of comfort. However, the latter individual had never used a computer before and

had also let his partner do all the work that involved using the prototype. Over half of the rest of the volunteer pairs used the prototype equally between them. Of those that did not, personal preference was the main reason for not doing so (64%), but also differing ability was another possible reason (36%).
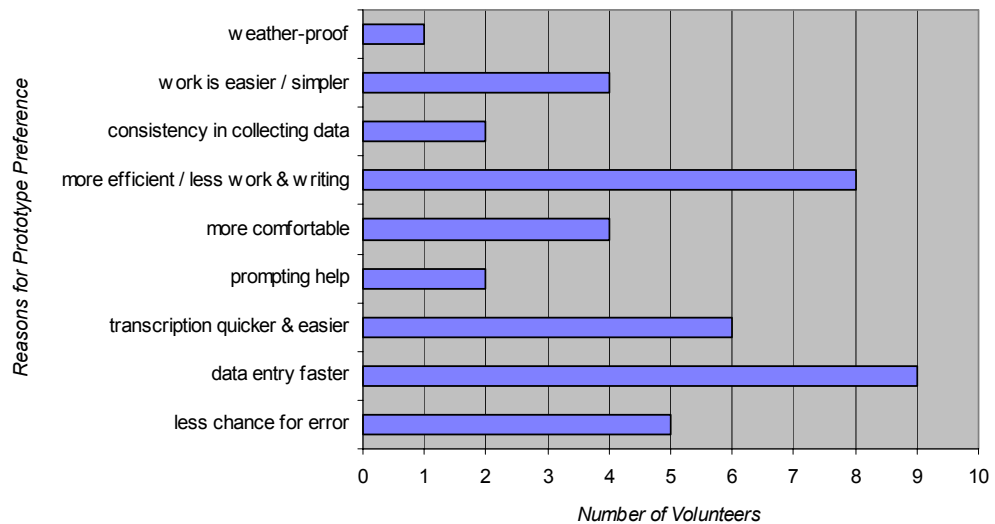
Figure 30. The reasons given by volunteers for preferring the prototype fieldwork tools over traditional pen and paper methods of data collection.

We also enquired as to whether the computing terminology was confusing to them. Everyone understood the computing terms, though there was some difficulty in some of the ecological terms which had been mistaken for computing ones!

We can safely conclude that the prototype system was well received by the volunteers, with the majority of people feeling comfortable in using the system after the half hour training period, and everyone expressing their preference for the prototype system over a manual alternative (even the individual who never mastered the system!).

5.4.3   Before and After Impression Analysis

Although the acceptance analysis shows that everyone preferred the prototype system over a manual one we also wanted to examine the volunteer's impressions of the prototype fieldwork tools before and after using them for the two weeks, and to see how their opinions changed after they had used the prototype. We did this by asking them to rate how they felt about using the PalmPilot before they arrived and how they

would feel about using it again on another project afterwards. We allowed them to express five levels of enthusiasm: looking forward to using it a lot, looking forward to using it a little, not worried one way or the other, a little worried about using it, very worried about using it.

In retrospect it may have been more rigidly correct to conduct the "before feelings" part of the questionnaire at the start of each volunteer groups' two-week stay. However, we have confidence in the integrity and memory of the volunteers in accurately reporting their feelings of two weeks before.

Before using the equipment the largest of volunteers (43%) were not worried about using the equipment one way or the other, a quarter of the group were looking forward to using it a lot, and the remaining quarter were equally divided between a little worried and looking forward to using it a little. No-one was very worried about using the equipment. After using the prototype for their two weeks the majority of volunteers were looking forward to using it again a lot (60%), some were looking forward to use it again a little (18%), and some were not worried one way or the other (23%). No-one was even a little worried about using the prototype after having used it for the two weeks. Forty-three percent of the group had not changed their minds about their views of the equipment, but these were the individuals with positive opinions to start with. There was a massive increase in the people looking forward to use the equipment a lot (an increase of 34%) with people who had the lowest enthusiasm originally now showing the largest positive swing in opinion. The before and after impression change is illustrated in Figure 31.
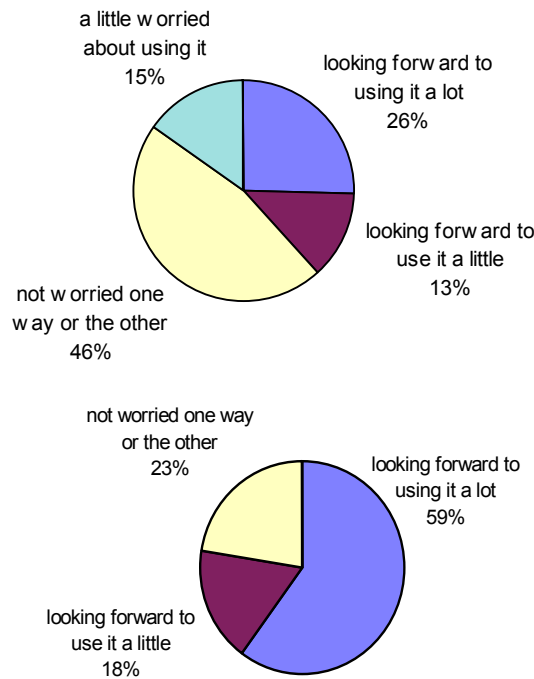
Figure 31. Volunteers feelings about the prototype before (left) and after (right) using it.

Before using the prototype the majority of volunteers were not worried one way or the other about using it, though a few were worried about using it. Afterwards the vast majority are really looking forward to use it again, and absolutely no-one is worried about using it at all. The majority of volunteer's perceptions were improved by at least one point on our five level scale.

We had only hoped for a "not worried one way or the other" level of enthusiasm. After all, the volunteers are there out of an interest in Africa and ecology not in computing equipment. It is therefore very gratifying to see that the majority of volunteers are now looking forward to using the prototype again, and it is certainly a good indicator of the prototypes success in the field.

5.4.4   Ease of Use Analysis

Although people's general impressions of the equipment were good we also probed them with more specific questions about the prototype's usability. In determining the ease of use of the prototype we considered both the PalmPilot system and our

software, and rated ease of use on a five level scale of: very difficult, difficult, okay, easy, very easy.

The PalmPilot provided two methods of data entry: the Graffiti hand-writing recognition system or an onscreen keyboard. The volunteers could use whichever method they felt most comfortable with, and switch between them as they wished. Of those who used Graffiti the vast majority (90%) found it ranged from okay to very easy to use, and the same applied for keyboard users. Only three people had any difficulty using Grafitti, and only one had any difficulty using the onscreen keyboard.

In using our prototype software, the selection of templates and general entry of data (two of the most important aspects of the fieldwork tools) were both found to be largely easy to very easy to use, with the remainder of volunteers finding them okay to use. There were no difficulties at all in either of these processes. The same pattern was found in a general ease of use rating for the prototype as a whole that the volunteers were requested to give.

We combined all of the ease of use results (both those of the PalmPilot and our software) to create a total ease of use rating. As shown in Figure 32, the vast majority of users either found it easy or very easy to use the prototype, with only a very small proportion having any difficulties.
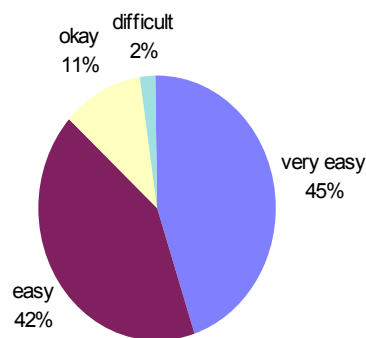
Figure 32. Overall ease of use rated by the volunteers on a 5 point scale from 'very difficult' to 'very easy'.

In order to be a little more specific in our usability analysis we asked the volunteers to state what data entry features they found particularly easy to use. The entering of GPS locations (stated by 39% of the group) and picklists (25%) were reported to be particularly easy to enter, the others are illustrated in Figure 33. We also asked users to

state any specific difficulties in entering data. Only eleven people reported finding any data particularly difficult to enter (six found text entry to be hard, two found entering numbers difficult, one had problems with entering a non-current time, and another individual had problems with entering the location - due to a faulty GPS cable).
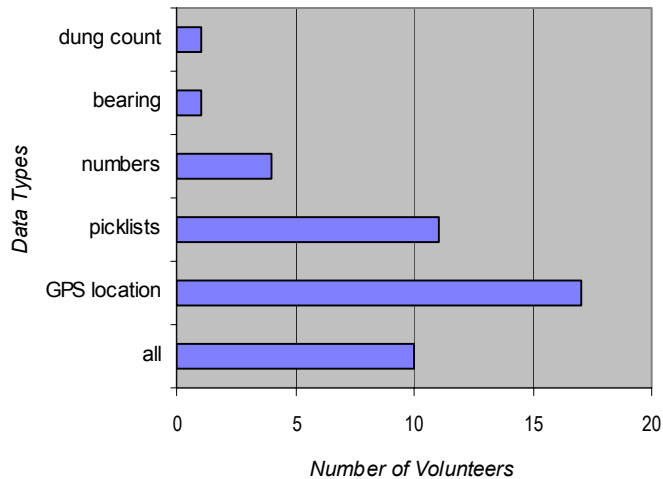
Figure 33. Types of data that were reported by volunteers as being particularly easy to enter.

The ease of use of the prototype has proven to be more than satisfactory for most users, with only a small number of users experiencing any difficulty (and those mainly due to the PalmPilot itself and not our software). The entering of data via a connected GPS receiver was the single most highly rated feature that users perceived as enhancing usability.

5.4.5   Likes and Dislikes Analysis

We asked the volunteers to state any particular likes of dislikes they had about the prototype, and to also comment on any suggestions they may have for improving the system. The four main likes about the prototype were the compact size of the device making it easy to carry around, the ease of entering data into the form display, the prompting of the type of data to enter, and the ease of downloading the data at the end of the day. These and all the other likes can be seen in Figure 34.
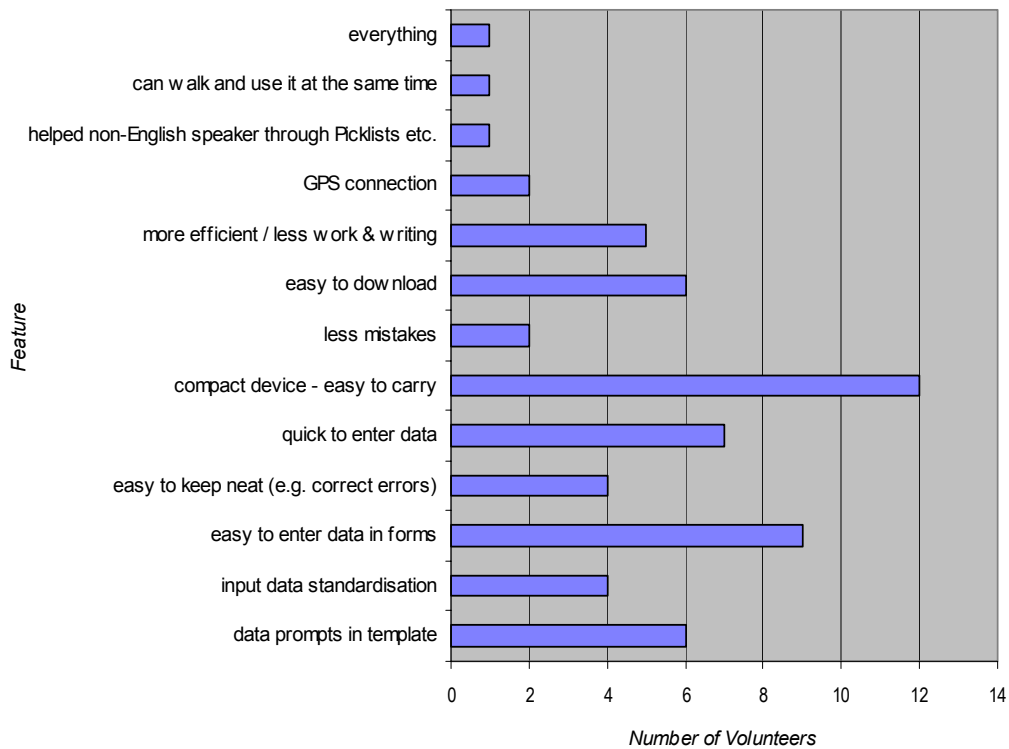
Figure 34. Features of the prototype liked by the volunteers.

The main dislike by far, expressed by half of the group, was the need to occasionally reset the device. This problem was caused by a bug in the software which has now been rectified. The only other major dislikes of significance (i.e. a dislike expressed by more than two people) was that the entry speed was sometimes too slow (expressed by 19% of the group) and the screen sometimes hard to read (16%).

The main suggestion for improvement was a better picklist (e.g. multiple selections, quicker access to it, etc.) followed by the suggestions of a less cumbersome PalmPilot-GPS cabling arrangement and an increase in the speed of the software. These and the other suggestions are shown in Figure 35.

There were double the amount of likes suggested as compared to dislikes, and over half of the dislikes have now been eliminated with the rectification of the reset bug. The remaining dislikes are mostly one-off opinions expressed by individuals rather than a consensus of opinion, bar the need to speed up the prototype and provide a better cabling arrangement.
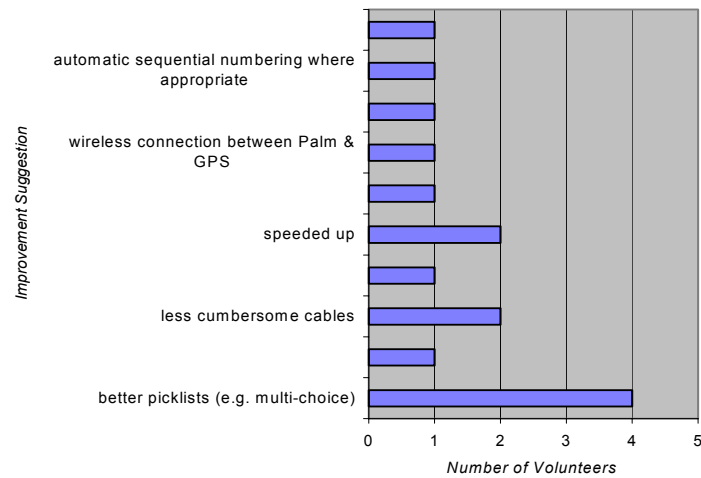
Figure 35. Suggestions for improvements to prototype made by volunteers.

5.4.6 Log Book Analysis

In addition to the questionnaire data, for the last two groups of volunteers we also kept a daily equipment log in which we recorded the equipment used, the number of resets required, any faults experienced, and the number of notes recorded. The complete log data is included in appendix H.

In examining the logs we can see that for the last two groups (of two weeks duration each) over four thousand stick-e notes were recorded, with eighty-eight resets incurred in the process. Although the resets were obviously frustrating to the users (though from the questionnaire data it does not appear to have impacted much on their enthusiasm for using the equipment) they never resulted in any data loss or other harmful side-effects. The only cause of serious problems (resulting in a backup paper notebook having to be used on nine occasions and the loss of some data on two occasions) was in the logistics of keeping the PalmPilot batteries charged. There was also one incident where there was a fault in linking a GPS receiver to a PalmPilot, which was traced to a faulty connector. We also asked for details of any other problems in the questionnaire, but none of any significance was presented other than the battery, reset, and speed

issues (the last of these having no negative effect on their work other than in limiting the rate at which data could be recorded).

5.4.7 Usability Study Summary

The usability study was conducted through a questionnaire presented to each member of a diverse group (in terms of age, sex and previous fieldwork and computing experience) of thirty-eight ecology volunteers. The acceptance of the technology was unanimous with everyone preferring the prototype over traditional data collection methods, and most of the group feeling comfortable with the system in the first day of use. There was a small degree of worry about using the prototype before trying it, but after their two-week projects all worries were dispelled and most people were in fact looking forward to using the system again given the opportunity.

Very few people had problems using the PalmPilot itself and no one had any difficulties in using the prototype software. One feature in particular that was identified by users as enhancing the ease of use was the automatic recording of location data. The overall usability of the prototype, rated on a five point scale ranging from very difficult to very easy, was found by the vast majority of people to be easy or very easy. The general features that user's said they liked most about the prototype centred on convenience: compact size, ease of data entry, the prompting of data to enter, and the ease of transcription. The only negative aspects were the resets required, the wish for an increase in speed by advanced users, and the battery charging logistics. The former two problems would be fixed in production quality software, and the latter could be rectified with self-charging internal batteries or simply better organisation of the charging process.

Given such positive feedback from the volunteers throughout the summer of trials we can confidently state that the usability of the tools has been a great success. However, these people are only one part of the equation: the ecology researchers must use the data collected from the tools, and also collect data with the tools themselves. We were already familiar with their very positive opinions on our tools from our early prototype work with them. Their interest in the tools led to this large field trial in the first place (they had requested to use our new tools to aid in data collection with this new volunteer-driven ecology project). They were pleased that, using the tools, the volunteer

data was indeed reliably and consistently collected and that it was available for immediate analysis after a simple download procedure.

5.5   Summary

We have performed a comprehensive usability study of the prototype in the field with a group of thirty-eight unbiased and diverse fieldwork volunteers over a period of eight weeks. In this chapter we firstly presented the structure of the fieldwork: pairs of volunteers performing various tasks over the course of a two week period, with teams of approximately ten people for each two-week period. We also described the diverse range of activities that each pair would be expected to carry out, ranging from rhino observation to elephant dung counting, and how the prototype would enable such activities through the recording of stick-e notes.

The benefits to the ecology project managers were clear from the outset: they would get immediate and effortless transcription of the recorded data into a spreadsheet (through a simple downloading procedure), the time and location data would be reliably recorded (because it was automatically extracted from a GPS receiver and system clock), the data collection would be performed in a consistent manner (via standard stick-e note templates and enforced data types and ranges), and the data could be easily secured (through various electronic backup strategies). However, we were also interested in how the prototype would be received by the volunteers actually performing the fieldwork. Therefore, to ascertain the usability of the tools in the field we gave each volunteer a questionnaire that aimed to capture their feelings and experiences of using the prototype. The results of an analysis of the data gathered were very positive: every single volunteer preferred the prototype to traditional methods, in particular citing features such as the ability to automatically record location data. Most users were comfortable using the prototype within the first day after only half an hour of training, and the majority were looking forward to using the prototype again in future projects. In conclusion, we can safely state that the prototype is certainly most useful, practical, and well received in real fieldwork projects.

Chapter 6:
Reconsidering Context and
Frameworks

The usability study effectively demonstrated the ability of context-aware systems to help people in real world tasks and situations. However, it is rare to find such context-aware systems in use today due to the large development overhead incurred in imbuing software with a context-awareness. In effect there is a need for generic context-aware services that can be compared to the GUI services available for user-interface developers; imagine developing a user interface by manipulating individual pixels on the screen. As was the case with the development of common GUI services, we believe that the construction of context-aware services will encourage the adoption and an increasing sophistication of context-awareness.

The stick-e note framework was our first attempt at providing a context service, and the fieldwork tools were the first applications developed to utilise this service. In this chapter we describe the lessons that we have learnt from the development of this software, and re-examine our beliefs about the nature of context and the facilities a context-aware framework should provide. We go on to present the general concept of a *context information service* and define its structure in terms of a layered model of context-aware capabilities. Using this model we compare a number of research projects that are exploring context-aware services, and conclude with some suggestions as to the desirable characteristics that a good context information service will exhibit.

6.1   The Nature of Context

At the start of our work on context-awareness we had conceived of context as being a relatively simple snapshot of a set of environmental values such as location, temperature, time, etc., which described a particular instance of the user's environment.

But as our work has progressed it has revealed a more rich and complex nature of context.

6.1.1 Context and Content May Not Be So Different

In our stick-e note framework we initially made a very clear distinction between context and content. This was understandable given our concepts of Post-It notes and tourist guides in which an item of data was electronically attached to a particular situation. However, in developing the data collection tools for the ecologists the difference became much less distinct. When using the tools they were just as interested in the context of the note (i.e. the location in which it was recorded) as they were in the behavioural observations that they recorded. Additionally, the information they recorded, such as the current activity of a giraffe, could be considered to be as much context as it was content. To them the only distinction between the content and context of a stick-e note was that the computer automatically completed the contextual parts.

We thus adjusted our concept of a stick-e note to accommodate this change of perception. Instead of two separate context and content parts, there is now simply a set of fields that describe a situation.

6.1.2 Context Can be Complex

Upon first consideration a context element such as location may appear to be a relatively simple item of data to manipulate. However, it soon becomes apparent that there is a plethora of different location sensing systems available such as GPS [Texas-University 1999 - #141], active badges [Want, Hopper 1992 - #145], visual identifiers [Rekimoto and Ayatsuka 2000 - #110], etc. These all tend to use their own measurement systems and work in different, though sometimes overlapping, domains. For example, GPS only works outdoors and tends to use latitude and longitude measurements, whereas active badges are more often deployed indoors (due to infrastructure costs) and use a cell-based measurement system. What is more, some of these individual measurement systems have different operational attributes that can dramatically affect their correspondence with the real world, e.g. latitude and longitude is expressed in relation to a specific datum (essentially, a mathematical representation of

the Earth's surface) – using the wrong datum can result in inaccuracies of as much as a kilometre. To further add to the complexity locations may be specified in a variety of ways, such as a point, a rectangle, or a polygon, each of with may have some form of accuracy or error boundary.

Providing comprehensive support for a context element such as location, considering the diversity of sensors and measurement systems, is a complex task.

6.1.3   Context is Much more than Location

Given the inherent mobility of handheld computers and the varying situations of ubiquitous computing, it is not surprising that the location context element has been the focus of much context-aware research. Indeed, our research was initially focused on this context element as there were a wide array of automatic detection mechanisms and obvious uses for location information. However, there are many other types of context element that can be automatically detected, some of which may be more important than location, depending, of course, on the application. For example, the orientation of the user may be just as important as location in trying to determine what they are currently looking at, and in our continuing work on the fieldwork tools we are now exploring contexts as diverse as time of day, weather conditions and animal footprint measurements.

6.1.4   Context is an Attribute not an Entity

In our initial stick-e note framework we considered context elements as individual entities that existed in the user's environment, e.g. the location, the temperature, etc. However, context really makes no sense on its own. It inherently describes some real or virtual entity. For example, a location may be attributed to a place, a person, a car, a book, etc. A location on its own has no value unless we know what entity it is locating.

In the stick-e note model where we thought of current context elements as self-contained states existing in the environment, what we were actually doing was making implicit assumptions about what they were attributed to. For example, the current location context element was implicitly attributed to the user, temperature was implicitly attributed to the local vicinity, orientation was implicitly attributed to the user,

giraffe behaviour (in the fieldwork tools) was implicitly attributed to the last giraffe spotted, etc. Such implicit assumptions may work in simple applications but they are not advisable. In more complex applications that consider the current context of more entities than simply the user and the immediate vicinity, it is essential that context elements be explicitly modelled as an attribute of a particular entity.

6.1.5 Virtual Entities can have More Complex Context

The current context of a physical object, such as a person, can be represented by a set of otherwise unrelated fields, e.g. their location, temperature and orientation. The context of virtual entities may be represented in the same way. For example, a virtual billboard could be associated with a real location context element so that the advertisement or information it presents can be displayed on a user's handheld computer when they walk by the location to which it is virtually attached. Unlike a real billboard though, the virtual billboard may be easily moved to different locations. The most interesting aspect of the virtual billboard however, and all other virtual objects, is that they need not obey the laws of physics as real objects do. The virtual billboard can therefore do otherwise impossible things like be in two or more places at the same time, be visible only to certain types of people, exist only at certain temperatures, and so on. The aforementioned attributes could be practically employed in a virtual billboard that advertises an ice-cream parlour, where the billboards could be displayed on any ice-cream lover's handheld computer when walking near the store on a hot day!

Given their unique ability to defy the laws of physics, virtual entities need to express their context in more complex terms than a simple set of fields. They require the ability to represent their context in the form of conditional expressions of context that may utilise Boolean operators.

6.2 The Nature of Context-Aware Applications

Given our comments on our evolving understanding of context, it is perhaps not surprising that we found context-aware applications to be much more diverse in structure and more time-consuming to develop than we had first anticipated.

6.2.1 There are Many Classes of Context-Aware Applications

The stick-e note framework suits applications that seek to place data in context and automatically retrieve that data when the user enters that context. However, our work on the context-aware fieldwork tools illustrates other types of applications, and there are no doubt many more new and existing applications that require other forms of context-awareness beyond what is offered by stick-e notes. This will particularly be the case when imbuing existing applications with context-awareness because the developer will probably prefer to add context-awareness in very specific places, in order to add some extra 'intelligence' to the program, rather than completely rewriting the application to fit the stick-e note framework. For example, a simple calendar and to-do list program could be imbued with a context-awareness so that it knows what context the diary and to-do entries will be carried out in. The to-do list could then offer a "do now/here" view and the diary could have a more intelligent alarm system than, for example, simply beeping half an hour before a meeting even when the user is currently in a completely different city to that of the meeting.

The stick-e note framework does indeed provide useful support for a certain class of context-aware applications, but there are no doubt other general classes of context-aware applications and there are certainly applications that have their own esoteric uses of context-awareness.

6.2.2 Context-Aware Applications tend to be Resource Hungry

A device or program that is aware of its context must, by definition, keep watch over its environment. In its simplest form this may be performed by occasionally polling some sensors that provide the current value of a particular context element. All context-aware applications need to perform this task, so if more than one context-aware application is simultaneously running on a device then there could be a competition for sensor resources; there is then a danger of one of the applications locking out the others. This is especially likely for sensors that require a continuing dialogue rather than periodically broadcasting information.

For many applications obtaining the current context from a sensor is just the start of its work. For example, in the stick-e note framework the current context is obtained in order for the triggering engine to check if there are any stick-e notes that should be

triggered in those circumstances. This continual triggering process, and similar processes in other context-aware applications, requires considerable processor resources and could again place the application in competition with other programs running on the device (including non-context-aware programs).

6.2.3 Context-Awareness has a High Development Cost

Writing a context-aware application from scratch is a hard process. To imbue a device or program with a knowledge of just a single context element requires a good deal of effort. One must develop the code to communicate with some external sensing system, convert the obtained data into a desirable format, provide the necessary machinery for manipulating the data, and finally, provide an interface with which the user can work with the data. In the case of imbuing the fieldwork tools with a location-awareness, we developed the code to communicate with a GPS receiver by listening to its broadcasts of NMEA sentences (a data transmission format commonly used by GPS receivers), provided functions to convert the latitude and longitude to other formats, e.g. UTM and OSGB (these are two co-ordinate systems that we commonly use), provided the means to calculate distances between locations, and provided a map-like visualisation of the geo-referenced forms. In other words, a large portion of the development cost was in producing the location-awareness capability.

This development effort does little to encourage the growth of the use of context-awareness in applications, especially in cases where the context-awareness is incidental to the main purpose of the application. For applications like the previously mentioned context-aware to-do list and diary, it is unlikely that the development cost could be justified for the relatively small benefits derived.

6.2.4 The Computing Environments are Diverse

During the relatively short time span of this work, approximately three years, we have seen many advances in the mobile computing platforms available, each offering ever more sophisticated computing capabilities. And, unlike desktop computing, the world of handheld and ubiquitous devices is a very heterogeneous one, with many different computing platforms each with their own esoteric operating systems.

Although this diversity makes for a quite exciting environment to work in, it does present a major drawback in developing new software: we can find ourselves spending more time rewriting the same programs for the latest platform than we can in developing the new ideas that we wanted to explore.

6.2.5 The Greater the Context the Greater the Application

Our work has primarily been based on handheld computers that we have equipped with sensors to detect various aspects of the immediate environment. This provides a user-centric context-awareness in which the device shares the same contextual viewpoint as the user, e.g. their location, the temperature there, etc. The more types of context we support, the better a perception of our context the device has. But perception can also be improved by incorporating the viewpoints of others, i.e. sharing contextual information with other devices to build a much more expansive contextual model of the world. Applications then have an enormous wealth of contextual information to utilise, not just about the user's context but about the context of the many other people, devices, and objects that are sharing that environment. Obviously much more sophisticated applications can be developed given such a dramatic increase in the richness of context.

6.3 The Need for a Context Information Service

The issues we have addressed thus far point to a common theme: that developing context-awareness is far from straightforward. In the current state of affairs incorporating a context-aware capability into software could be likened, as we have already mentioned, to developing a program's graphical user interface by manipulating individual bits and pixels rather than using the high-level facilities provided by a common interface environment such as X-Windows or Microsoft Windows. One can easily imagine that today's user interfaces would be substantially less sophisticated without the facilities provided by these environments. It is our belief that just as generic interface services have enabled the growth and improvement of the graphical user interfaces of software, so too will generic context services support the growth and improvement of the context-aware capabilities of software. The context toolkit project at Georgia Tech [Salber, Dey 1999 - #120] offers the same vision, though they focus on

developing the equivalent of individual GUI widgets whereas we concentrate on developing the equivalent of the shared windowing environment in which such widgets can be developed and made available to applications and end-users. Work in both areas is essential and complementary.

We believe that at the core of a supporting infrastructure for context-aware computing should be what we have termed a context information service (CIS) [Pascoe 1998 - #98] [Pascoe, Ryan 1999 - #102]. The responsibilities of this service are to gather, model, and provide contextual data, i.e. the general functionality that is required at the heart of all context-aware applications. Unlike the stick-e note framework, the CIS does not attempt to impose any application semantics: its role is in the provision of contextual information. It is the common base on top of which more sophisticated context-aware capabilities and applications may be built.

Most of the work on developing context information services originates from groups that have been involved with specific applications or contexts, and who have sought to develop a general CIS to support their efforts. This tends to result in CISs that have biases or specialities, or that are tailored for a particular job. Each offers their own unique contribution and some degree of generality, but we do not believe that a sufficiently general-purpose service has yet been developed to support the full potential range of context-aware applications. In an effort to rectify this situation the remainder of this chapter explores the design space of context information services. We hope that this will clarify the desirable characteristics of such services and thus help in the construction of better services and, in turn, assist in building better and more sophisticated context-aware applications.

With the aid of an examination of a wide range of different context-aware applications and services, we have segmented the CIS design space into five layers that represent common logical groupings of purpose or activities, as shown in Figure 36. The figure also illustrates that a CIS could be distributed at the granularity of either complete layers or sub-sets of a layer.
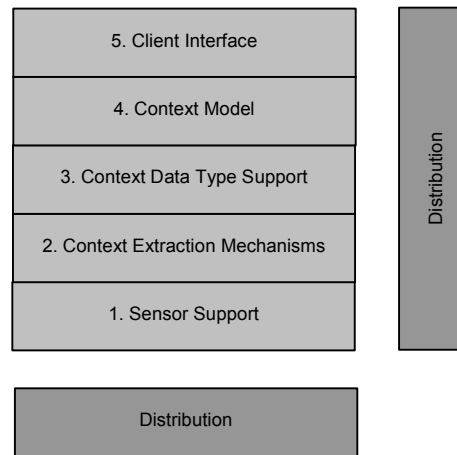
Figure 36. Layers of a Context Information Service.

We examine each CIS layer in remaining sections of this chapter. The potential characteristics of each layer are enumerated and the pros and cons of different design options are discussed. To aid in this discussion we have selected a diverse set of CIS research projects (chosen to represent the current state-of-the-art of CIS research) that we can examine to better understand the possible design options. A table of characteristics is presented for each CIS layer so that the different approaches of the reference projects can be easily compared. This should also be useful in providing an overview of what aspects of CIS design have been, or are being, addressed and which have yet to be fully explored. A brief summary of each of the reference projects is given below before progressing to the first CIS layer. We then relate these systems (using information taken from the cited publications) to our layers.

Application-Aware Adaptation (Odyssey)	This CIS developed at Carnegie Mellon provides some extensions to the UNIX operating system to enable application-aware adaptation in a mobile environment.	[Satyanarayanan, Noble 1994 - #124]
AROMA	Roskilde University are exploring how remote awareness of an office can be supported through a CIS that captures context from one place, abstracts it, and sends it to another.	[Pedersen and Sokolar 1997 - #104]
Context-Aware Framework	Schilit's context-aware framework was the first attempt at developing a CIS. It supports the modelling of various object's context but is primarily location based.	[Schilit, Adams 1994 - #125]
Context Toolkit	Georgia Tech have developed a broad range of context-aware applications and are now focusing on developing context-widgets akin to user interface widgets in a GUI toolkit.	[Dey, Abowd 1999 - #29] [Salber, Dey 1999 - #120]
Environment Aware Mobile Applications	The objective of the CIS developed at Rutgers is to support the adaptation of applications based on information about their changing mobile environment and available resources.	[Welling and Badrinath 1997 - #151]
General Location Service	Leonhardt's CIS is focused entirely on location contexts, developing a unified model in which diverse location types and sensing technologies can be accommodated.	[Leonhardt and Magee 1996 - #80] [Leonhardt and Magee 1998 - #78]
Limbo	Lancaster University has been involved in context-aware projects with a strong networking element and has developed a context information service based on tuples.	[Davies, Friday 1998 - #25] [Davies, Wade 1997 - #26]
SAMoA	The Fraunhofer Institute for Computer Graphics have been developing a CIS to support the inherent context-switching of people with multi-tasking jobs.	[Chavez, Ide 1998 - #20]
SitComp	Hewlett-Packard (UK) Labs' CIS is based on companion and location contexts and explores how to interpret and abstract context in order to be more useful and understandable.	[Hull, Neaves 1997 - #64]
SPIRIT	The SPIRIT work at AT&T Labs UK (formerly ORL) focuses on providing information on people and hardware devices within a richly networked environment.	[Adly, Steggles 1997 - #4] [Harter, Hopper 1999 - #58]
X.500 Location Information Service	Philips Research Labs have developed a location-oriented CIS that uses X.500 remote directory access mechanisms to structure and access contextual information.	[Maaß 1998 - #86]

Table 20. Summary of reference projects chosen to explore the CIS design space.

6.3.1 Layer 1: Sensor Support

Underlying all context-aware applications is some form of sensor, be it hardware devices like GPS receivers and active badges, or software devices like user-logon sensors. These sensors are the point of first contact with the outside world and are responsible for the capture of raw context data.

	Fixed		Extensible	
	Single	Multiple	Off-line	At Run-time
Application Aware Adaptation		✔		
AROMA		✔		
Context-Aware Framework	✔			
Context Toolkit				✔
Environment Aware Mobile Applications				✔
General Location Service				✔
Limbo				✔
SAMoA		✔		
SitComp		✔		
SPIRIT		✔		
X.500 Location Information Service	✔			

Table 21. Sensor support.

The flexibility and potential scope of a CIS is very much dependent on its underlying infrastructure for sensor support. We offer a classification of four types to define the flexibility of sensor support, increasing in flexibility from a fixed single sensor, to multiple fixed sensors, to an extensible infrastructure that allows new sensors to be added while the service is offline. The ultimate in flexibility is the ability to add sensors at run-time without interrupting the CIS (useful for server-based CISs or any constant and long-running CIS).

Although any system could be considered extensible from a developer's perspective (i.e. by writing and adding the code for a new sensor) we only consider the sensor support

truly extensible if it facilitates this at the *client or user level*. That is, the CIS should provide an interface for the client or user to easily install and configure a new sensor module, e.g. akin to installing a plug-in for a Web browser.

The X.500 location service CIS and the context-aware framework CIS base their sensor support specifically on active badge or tagging systems, i.e. a single fixed sensor. However, the majority of CISs have multiple fixed sensors. The fixed nature of these sensors is more inherent in application-specific CISs like SAMoA which utilise a specific set of sensors to achieve a specific goal. More general services, such as SitComp or SPIRIT, have in fact taken into account extensibility but we still categorise them as having fixed sensor support because they are extensible only at a developer level rather than a client or user level.

Only four CISs have client-level extensibility. The Limbo CIS provides a tuple space into which anyone can inject a tuple of context data, making it very extensible but also lacking in any management or other support facilities for sensors. The general location service CIS provides a management API that can be used to extend sensor support (though only for location sensors) and the Environment Aware Mobile Applications CIS provides an infrastructure into which new factories (equivalent to sensors) can be added. The context toolkit CIS focuses on developing context widgets (equivalent to highly abstracted sensors) which exist in their own right rather than necessarily being part of a larger CIS. They are inherently extensible as new widgets can be installed anywhere on a network, but they lack an over-arching management infrastructure.

In summary, sensor support in CISs is generally limited to a fixed set of sensors and where extensibility is provided it is often at the cost of the lack of a comprehensive management support infrastructure.

6.3.2 Layer 2: Context Data Extraction

The raw data from sensors often needs to be processed in some way in order to transform it into a more useful or stable form for a CIS client. For example, the names of places are often more useful than latitude and longitude values. We have identified four general extraction mechanisms that may be employed within a CIS. The simplest is to merely pass on the raw sensor data. However, *abstraction mechanisms* can convert this

raw data into a more meaningful form, e.g. Latitude 0° 1' 41" Longitude 36° 54' 10" abstracted to the place name "Sweetwater's Research Centre". Often the output of these abstraction mechanisms or sensors can form an input into another type of sensor, so creating a mesh of interconnected sensors. We call this a sensor *chaining mechanism*, a term first used to describe this technique in [Abowd, Dey 1997 - #2]. For example, the outputs from a location sensor and a time sensor could form the inputs to another sensor that produced a tide-level context computed from the input values.

	Mechanisms				Q.O.S.	
	Direct Value	Abstraction	Chaining	By Reference	Client Driven	Model Driven
Application Aware Adaptation	✓					
AROMA	✓	✓	✓			
Context-Aware Framework	✓				✓	
Context Toolkit	✓	✓	✓			
Environment Aware Mobile Applications	✓					
General Location Service	✓	✓				
Limbo	✓					
SAMoA	✓					
SitComp	✓	✓				
SPIRIT	✓					
X.500 Location Information Service	✓			✓		

Table 22. Context extraction support.

Combining abstraction and chaining mechanisms provides a sophisticated infrastructure with which to process, transform, or fuse sensor data. These may be resource intensive processes so a *referencing mechanism* is useful, whereby the context of one object can be derived directly from that of another rather than having to be independently computed. For example, if I am travelling on a bus then my positional context will be the same as the bus's, so my PDA's CIS can simply reference the bus's location rather than working out my location itself.

The context extraction layer may also offer quality of service (QOS) facilities that enable the balancing of sensor cost (e.g. processor time, power, etc.) with context quality (e.g. accuracy, timeliness, smoothing of erratic data, etc.). The QOS may be negotiated directly by each CIS client or, better, from a shared model of context where the quality of service is derived from the characteristics of the objects being modelled. For example, consider several CISs interested in the location of a person named Fred. Instead of each CIS negotiating its own QOS for monitoring Fred's location (i.e. specifying how often to update his location) the type of object can help provide a default QOS. In the case of Fred we know that he is a person and we also know the way in which people generally change location (much different in speed and direction from that of, say, a car). We can therefore provide a default update rate for Fred's location context based on the way a typical person moves around. Of course, in this case we also have to take into account if the person is inside another object (e.g. travelling in a car), though assuming a default mode of changing location still be useful in many cases.

Many CISs such as the context-aware framework and SPIRIT utilise the context data generated straight from sensors or, as is the case with Limbo, have no part in extracting the context data at all (although it does support the propagation, and potential conversion, of tuples from one CIS to another). Some CISs provide abstraction mechanisms, e.g. the general location service CIS that represents location data in an abstract logical form rather than as co-ordinates or cell identifiers. Only AROMA and the context toolkit provide explicit support for chaining sensors together to generate new contexts, although the general location service CIS and SitComp CIS do provide facilities for combining or fusing sensor data of the same general type in order to generate more robust and reliable context data.

The context-aware framework is the only CIS that seeks to provide QOS functions to regulate the extraction of context data. The other CISs appear to extract data at a rate predetermined by the CIS or sensor device rather than the more economical method of extracting at a minimum rate to maintain a desired level of quality of service.

In general, CISs tend to provide data straight from sensors without much support for the regulation of the extraction process (i.e. QOS) or for adding value to that data via abstraction or sensor chaining.

6.3.3 Layer 3: Context Data Type Support

As with low-level sensors, the flexibility of context data type support has a direct influence on the potential capability of the whole CIS. The context data output from the extraction layer is stored and presented as one or more available data types. Context may also come from clients who supply their own context data manually, i.e. they will also require context data type support. For example, a user may decide to provide her handheld fieldwork computer with a knowledge of which giraffe she is watching by specifying its name using a custom-made "observing giraffe" context data type.

	Fixed				Extensible	
	Single	Application Specific	Limited Set	Diverse Set	Off-line	At Run-time
Application Aware Adaptation			✓			
AROMA				✓		
Context-Aware Framework	✓					
Context Toolkit				✓		
Environment Aware Mobile Applications						✓
General Location Service	✓					
Limbo						✓
SAMoA		✓				
SitComp			✓			
SPIRIT				✓		
X.500 Location Information Service	✓					

Table 23. Context data type support.

We classify CIS data type support into two broad categories: *fixed* (where the CIS is limited to the data types provided by the developer) and *extensible* (where the CIS can be extended with new data types). The fixed category data types can be subdivided into *single type* (only one data type provided), *application specific* (more than one data type but for very esoteric tasks), *limited set* (2-3 general-purpose data types), and *diverse set* (many

general-purpose data types) support. We divide the extensible data type support into *off-line* (where the service has to be stopped to install the new data types) and *run-time* (where installation can progress without service disruption). As with sensor support we mean extensibility at the client and user level rather than at the developer level.

It is not surprising to find that the data type support of CISs mirrors quite closely their sensor support, as it is normally not very useful to be able to add new sensors without also adding new context data types to support them. Exceptions are CISs in which only certain classes of sensors can be added whose generated values can be mapped onto the existing data types, e.g. the general location service CIS in which a new location sensor's values are mapped onto the existing location data types. Also, the context toolkit CIS intends to provide a diverse and extensible array of widgets to provide context data, but individually these widgets have fixed type support.

Although the context-aware framework CIS and X.500 have a single fixed context type, a string type, it can be used to represent all sorts of different types of data in textual form. Of course, this will not be as efficient as using specialist data types and will also not offer the other advantages of using specific types, such as preventing incompatible assignments, etc.

As with sensor support there is a general correlation in increasing extensibility with decreasing management infrastructure. However, we believe that both management infrastructure *and* extensibility are required. This is provided so some extent by the environment-aware mobile applications CIS, which allows new factories (the equivalent of sensors) to be added to a standard infrastructure. However, there is still no such infrastructure for context data type support: each factory may generate whatever data types it cares to.

6.3.4   Layer 4: Context Model

The context model of a CIS organises the individual context values obtained from underlying layers into a coherent and uniform model to be presented to a client. We have grouped the capabilities of such models into four categories: *structure*, *viewpoint*, *extensibility* and *service support*.

Some CISs may be relatively structure-less and have no high-level data model at all. They may instead simply provide an abstract interface onto the underlying sensors. Other CISs may have very application specific or esoteric data models. More general-purpose model support comes in the form of name-value pair structures, tuples, or a completely object-oriented model in which context values are expressed as attributes of an object representing a real or conceptual entity.

Many of the reference CISs provide simple context models or even no context model at all. This is because they are essentially acting as conduits through which sensor data flows. In CISs such as AROMA these conduits may perform quite complex conversion or interpretation processes but are nevertheless simply providing abstractions of underlying sensors rather than generating their own contextual model of the world. However, the context-aware framework, SPIRIT, and X.500 location service CISs all provide sophisticated context models in which context data is presented as attributes that describe particular objects.

	Structure					Viewpoint	
	None	Esoteric	Name/Value Pairs	Tuples	Object Oriented	Focused	Diffuse
Application Aware Adaptation		✔					✔
AROMA	✔					✔	
Context-Aware Framework					✔		✔
Context Toolkit			✔			✔	
Environment Aware Mobile Applications	✔					✔	
General Location Service		✔					✔
Limbo				✔			✔
SAMoA		✔				✔	
SitComp	✔					✔	
SPIRIT					✔		✔
X.500 Location Information Service					✔		✔

Table 24. Context model support (1).

A critical decision to be taken in the design of a CIS is the choice of model viewpoint. A *focused viewpoint* refers to CISs that model the context of only one particular entity (be it real or conceptual): for example, a CIS that is solely concerned with monitoring a single patient's physiological condition. Although suited to some specific applications this viewpoint is extremely limiting as only context information pertaining to one specific entity may be modelled. For example, the CIS would be able to monitor an office-worker's location but not the location of the surrounding office equipment. In contrast, models with a *diffuse viewpoint* enable the context of many different objects to be modelled simultaneously.

An object-oriented structure and a diffuse viewpoint make for a good combination. The object orientation provides the generic structure necessary to enable the construction of a diffuse viewpoint model that could contain any number and type of objects of interest. For example, the context of a whole office environment could be represented in a single model consisting of people, printers, and other objects of interest, with their associated context attributes.

There is a strong correlation between object-oriented structure and diffuse viewpoints and vice versa (i.e. simple structures and focused viewpoints) in the reference CIS projects. The CISs with esoteric structures are designed for specific tasks (e.g. the general location service CIS is designed explicitly to model location contexts) and fall equally into either diffuse or focused viewpoints. For example, the application-aware adaptation CIS extends the UNIX operating system with a number of context states that describe a number of different resources (i.e. providing a diffuse view of different resources) whereas the SAMoA CIS is concerned with the task status of a single individual (i.e. providing a focused view of a single person). The context toolkit CIS is quite unusual in that its context widgets tend to provide a model from the perspective of the sensors rather than the objects being sensed. That is, the model tends to be of a "what can I see here" nature rather than one that tries to construct a uniform context model of known or detected objects. Its model support lies midway between a simple sensor conduit and a comprehensive context model.

 An extensible model is essential for a general-purpose CIS as it is obviously not possible to provide a model that includes every conceivable object of the real world or imagination. The clients should, therefore, be able to easily add their own objects of

interest to the context model as and when required. As with extensibility in other layers of the CIS we distinguish between an off-line extensibility (where the service has to be temporarily halted to add the new support) and run-time extensibility (in which there is no service disruption).

Many of the reference CISs provide a static context model (if they provide a context model at all) that cannot be changed at the client or user level. It may be possible to add new *instances* of structures or objects to such models, but not to add new *types* of structure or object. For example, a static-model CIS that provided a person object type would be able to add new instances of people but would not be able to add a new type of object to represent, say, a giraffe. This limits such CISs to specific applications or the level of generality that the original designer envisioned.

Most of the CISs that can be extended can be augmented as and when necessary during their run-time. For example, a new object that models the location and state of printers could be added to the context-aware framework CIS at any time. However, all the CISs that provide such run-time extensibility do so by providing a general structure that can describe any type object, e.g. a set of name value pairs in the X.500 directory service CIS or a tuple with an object type attribute in the Limbo CIS. The X.500 directory service CIS also has the disadvantage that the clients must be aware of the directory organisation in order to use it.

None of the reference CISs provides a more sophisticated object model that can support extensible types of object or structure. However, such a model would indeed be useful in the management, organisation and validation of the context model. For example, a person object may have its location context managed in a different way from that of a vehicle due to the different speeds and ways they change, i.e. different classes of object may have different default quality of service thresholds. Knowing how the context a particular object type changes could also be used to validate sensor readings, e.g. a person would rarely accelerate from 0 to 60 kph in ten seconds when walking from office to office. It could also help clients to inquire about certain classes of object, e.g. people and printers.

The final grouping of context model characteristics is service support. This includes any value-added services provided to work with the CIS model other than simple context

accessor services. The general services we have identified include *relationship modelling* (e.g. establishing a near-to relationship between objects), *triggering* by context (e.g. notification when a person approaches the Cathedral), *manipulators* (e.g. methods for calculating distances between two locations), and *viewers* to visualise the context data (e.g. a town map display onto which the model's objects are superimposed).

There are few services present in the reference CIS projects other than viewers to display location data (as in the context-aware framework CIS and X.500 CIS) and some triggering mechanisms, which could alternatively be viewed as more complex forms of an event driven interface (see next section). Services tend to be either very basic (e.g. the relationship service in the X.500 CIS only allows tags to be associated with objects) or application specific (e.g. the manipulators in SAMoA are designed specifically for aiding task organisation). AROMA lies somewhere between the two extremes as it supports a diversity of context types but with the specific aim of providing a viewing service that displays context information in a peripheral fashion.

	Extensible		Service Support			
	Off-line	At Run-time	Relationships	Triggering	Manipulators	Viewers
Application Aware Adaptation				✓		
AROMA						✓
Context-Aware Framework		✓	✓			✓
Context Toolkit						
Environment Aware Mobile Applications						
General Location Service		✓				
Limbo		✓				
SAMoA				✓	✓	✓
SitComp						
SPIRIT				✓		
X.500 Location Information Service		✓	✓	✓		✓

Table 25. Context model support (2).

We believe that services are an optional part of the context model as the main role of the CIS is as an information provider. Services that utilise or manipulate this information can be constructed as additional layers of a context-aware architecture. Indeed, it is often preferable to do this in order to keep the underlying CIS as general-purpose as possible. The general location service does this by separating its ActiveMap location viewer from the underlying CIS. We are taking this service-separation approach in our own research by providing a triggering engine that will sit on top of our context information service. We envision context services existing in multiple layers of a context-aware architecture with a CIS at its base.

6.3.5   Layer 5: Client Interface

The top layer of the CIS mediates the interaction between CIS and clients, forming the access-medium to all the context information contained within the CIS. Therefore it is ultimately the design of this layer that is critical to the success of the CIS as a whole. We have grouped the design characteristics of this layer into communication *mechanisms* (i.e. the method of dialogue between client and CIS) and *channels* (i.e. the type of connecting 'pipe' through which the dialogue proceeds).

We have identified three mechanisms that the CIS interface may use as a means to communicate with its clients. The simplest mechanisms are *polling* and *streaming*. In polling the client can only receive context information by explicitly requesting the value of a specific context state. With a streaming mechanism the client continually receives a stream of data containing the most current contextual information. In such a scheme the client may be able to specify which contextual states are of interest and the rate at which the stream is sent. A more sophisticated means of communication is an *event-driven* mechanism that, in its simplest form, will update the client when a change in a context state occurs. More sophisticated forms of this mechanism will allow clients to specify what states they are interested in and to define the value boundaries under which an event is generated, e.g. let me know when I have walked another 50 metres. The latter case shares much in common with triggering and in our work we are considering combining the two as a separate layer outside of the core CIS.

Note that no single communication mechanism is preferable to the rest. One application may require a polling mechanism whereas another may be more suited to an

event-driven mechanism, i.e. the best mechanism is determined by the nature of the client. For example, a program that printed to the nearest printer would only need information on the context of printers when the user wanted to print (i.e. a polling interface), a speedometer program would require a continuous feed of context data (i.e. a streaming interface), and a program monitoring people in a meeting would be better served by events generated as people enter or exit the meeting room (i.e. an event driven interface). Therefore, a CIS should try to support as many of the communication mechanisms as possible.

Whereas the communication mechanism defines the type of dialogue between CIS and client, the communication channel defines the type of connection through which the dialogue proceeds. We have identified three types of channel. A *broadcast channel* provides a kind of 'information bus' onto which all clients are connected; when a client has been connected it is then able to listen to all the context information that is transmitted on this shared single channel. At the other extreme separate *client channels* may be used to allow the CIS to communicate with each client individually. The best solution is a *client group channel* that is a flexible combination of both of the aforementioned channel types. With client group channels the CIS can assign one or more clients to one or more client group channels, facilitating the effective rationalisation of context information transmission. For example, instead of having separate channels to each client interested in the location of a person named Fred, many clients could be assigned to a single 'Fred channel'. Then instead of communicating Fred's location several times on a number of separate channels the CIS need only do this once on the single client group channel.

The final characteristic of the client interface is whether it provides any *low-level hooks*. These are useful to developers who are building context infrastructure on top of the CIS, e.g. building a triggering service. They provide access to critical parts of the CIS in a faster and more efficient fashion than may be possible if using the client interface.

The majority of the reference CISs that are intended as general CISs provide both polling and event driven interfaces, allowing them to cover a diverse range of applications. The application-aware adaptation CIS and the environment-aware mobile applications CIS only provide an event driven interface as they are designed as extensions of existing event mechanisms of operating systems. Although AROMA is

the only CIS to explicitly support streaming (its primary role is providing real-time views of remote contexts), event driven interfaces can simulate a streaming interface by generating suitably fine-grained events.

	Mechanisms			Channels			
	Polling	Streaming	Event Driven	Broadcast	Client	Client Group	Low-level Hooks
Application Aware Adaptation			✔		✔		
AROMA		✔					
Context-Aware Framework	✔		✔			✔	
Context Toolkit	✔		✔		✔		
Environment Aware Mobile Applications			✔			✔	
General Location Service	✔		✔		✔		
Limbo	✔				✔		
SAMoA			✔		✔		
SitComp	✔		✔		✔		
SPIRIT	✔		✔		✔		
X.500 Location Information Service	✔		✔		✔		

Table 26. Client interface characteristics.

Nearly all the CISs have dedicated channels to each client, with only two CISs allowing the efficient grouping of clients on a shared channel. The environment-aware mobile applications CIS provides a limited predefined grouping based on classes of events, e.g. a channel dedicated to events related to network connectivity. The context-aware framework provides the most flexible and dynamic channel allocation by dynamically grouping clients on channels that relate to a common interest. Note that AROMA has no channels as it doesn't have clients as such but rather internal viewer services.

Although no low-level hook support is indicated for any CIS, some of the CISs may actually support them - such information isn't readily available from the literature.

6.3.6   Distribution

Distribution does not feature as a single layer in our CIS model because, as is the case with many systems, there are any number of areas within the CIS that could be potentially distributed. Indeed, the distribution could occur within any of the layers or even by distributing the layers themselves. However, three areas of particular importance are the distribution of:

➢ *Sensors*, e.g. a remote weather sensor.

➢ *Context model*, e.g. transparently combining the context model of myself, held on my handheld PDA, with the context model of my house, held on my home computer.

➢ *Processing*, e.g. different computers responsible for the updating of different parts of the context model or for the provision of different services.

CIS designs have approached distribution in different ways ranging from it being an essential characteristic of the CIS to not being supported at all. This has a direct impact on the type of computer systems that the CIS can run on because if distribution is essential to the operation of a CIS then a permanent network connection must be available (ideally at all times, although some permanent distributions may be able to sustain a limited period of disconnection). For highly mobile devices such as PDAs, in which a CIS could be most useful, the luxury of a permanent network connection is often just not possible let alone desirable. So permanently distributed CISs are generally limited to locales that have a high-level of readily available infrastructure to support them. This is the case with the SPIRIT project, which provides a permanently distributed CIS in a highly networked office environment.

In between these two extremes of permanent and no distribution lies opportunistic distribution. This is the most flexible method of distribution, where the CIS can operate with no network connection at all but can also utilise distribution if the resources become available. For example, the opportunistic distributed CIS could utilise a weather monitor unit that it discovers nearby as a new sensor, or it could expand its contextual model with the positions and times of some buses discovered from a CIS at a bus stop.

	Sensors	Context Model	Processing
Application Aware Adaptation	-	-	-
AROMA	O	-	P
Context-Aware Framework	P	O	P
Context Toolkit	P	-	-
Environment Aware Mobile Applications	-	-	-
General Location Service	O	O	O
Limbo	-	O	-
SAMoA	-	-	-
SitComp	P	-	-
SPIRIT	P	P	P
X.500 Location Information Service	P	P	-

Table 27. Distribution characteristics (P = permanent distribution, O = opportunistic distribution, '-' = no distribution supported).

The reference projects present the full range of distribution options ranging from completely stand-alone non-distributed systems such as SAMoA to the permanent and complete distribution of CISs such as SPIRIT. Distribution of sensors is the most common form of distribution, with CISs such as the context-aware framework and SitComp using distributed tagging technologies, and the AROMA CIS being able to incorporate many types of networked device (e.g. video cameras and microphones) as sensors. Distribution of the context model and processing is less common. The Limbo CIS, the general location service CIS, and the context-aware framework CIS, all allow their context models to be fragmented and distributed across a network, whereas the context models of the SPIRIT CIS and the X.500 directory service CIS actually insist on such a distribution. The distribution of processing is possible with the general location service CIS and is inherent in the remaining CISs that support it, e.g. the SPIRIT CIS whose processing utilises distributed components such as an object-oriented database and various CORBA objects.

We believe that CISs supporting distribution in an opportunistic fashion are likely to form the best general-purpose services as they work equally well in connected and unconnected environments, exploiting the currently available resources to provide as good a service as possible.

6.4   Summary

In this chapter we have presented our views on the nature of context and context-aware applications, views that have evolved with experience of developing context-aware software. We have found context to be much subtler and complex than we had originally proposed. Differences between context and content are blurred, working with context elements may involve sophisticated manipulations, and each element inherently describes some real world or virtual entity.

This more sophisticated view of context-awareness calls for some fundamental context services in order to support its implementation. The need is for low-level common services that provide support beneath the level of any application-specific semantics. We have proposed that services fit into a five-layer model that addresses sensor support, context data extraction mechanisms, context data type support, some form of context model, and the type of client interface provided to the services offered. In order to explore the design space of each service layer we compared a number of research projects that have attempted to address the issue of context service provision.

In our survey of context service projects we found none that completely succeeded. However, based on the suggestions we made for each service layer, in the next chapter we propose our own model implementation of a prototype CIS that does attempt meet all of our needs. Then in the final chapter of the thesis we explore how we might distribute such a context information service.

Chapter 7:

A Prototype CIS Design

In this chapter we present a prototype CIS design that attempts to address the issues and guidelines that have been raised in the previous chapter. The general aim of the CIS is to offer the basic context provision service on top of which more complex context services (such as stick-e notes) and applications may be constructed.

The CIS is a computer program's first point of contact with the external world. As such its duty is to discover information about the current context and present it to clients in a useful and understandable form, providing a portal onto the state of the real and imaginary world around us. The CIS must be simple for clients to use (our goal after all is to encourage the use of context-awareness through these services) and at the same time be flexible enough to support a plethora of different types of context element.

The design of the components and interfaces of the CIS we describe in this chapter is complete, but a full implementation of the service has yet to be realised (due to time constraints, not feasibility). We have succeeded in building a prototype of all the modelling components but have yet to implement the service components (the two types of components are described in the following paragraphs).

7.1 CIS Component Overview

Our CIS design is comprised of a number of modular components that a client may assemble and connect together in various ways. These components can be divided into two categories: (i) the *modelling components* that can be used to build a contextual model of the world, and (ii) *service components* through which this contextual model may be constructed, manipulated, and viewed. We start by describing the modelling components.

The context modelling components enable a client to build an object-oriented model of the subset of the real or imaginary world that is of interest to them. It also allows a client to link the model to a sensor system so that it may be constantly updated. Consider, for example, a context-aware shopping list program running on a palmtop computer. Its task is to remind the user of the things she needs to buy in each shop that she walks by. Such a program is interested in modelling the subset of the real world comprised of the location of shops and its end-user, and perhaps also the current time and the opening times of the shops. Using the modelling components of the CIS a model consisting of a number of shop objects and a user may be quickly constructed and linked to a location sensing system in order to keep the model continually updated.

The CIS provides five types of modelling components, as illustrated in Figure 37. To the far left of the diagram is an *artefact*, which is a generic structure that can be used to represent a real or imaginary physical object, such as a person. The artefact may consist of a number of state components. Each state of an artefact represents the value of a particular context element, such as the location of a person.
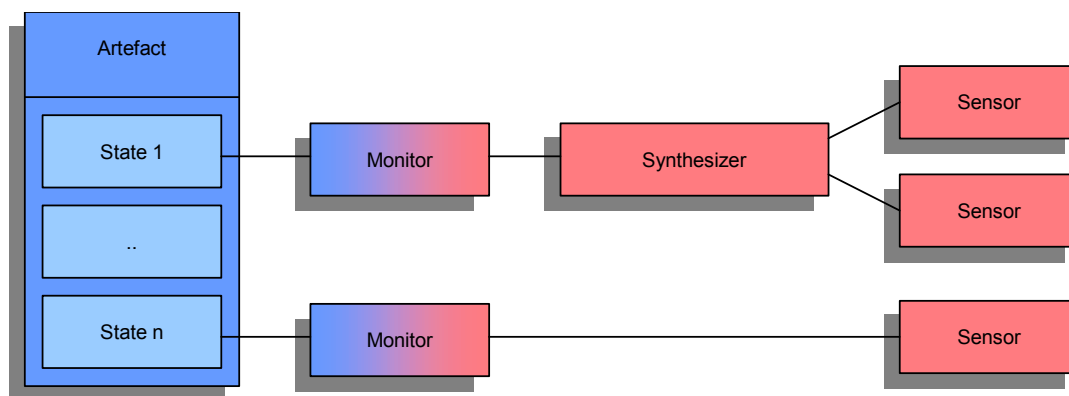
Figure 37. Modelling components of the CIS.

To the far right of the diagram are a number of sensor components, which interface with a physical or software-based sensing system and provide current context element values obtained from a sensor. There is a one-to-one mapping between a sensor component and a physical sensor system, but the current values of one or more sensing systems may be combined with a *synthesizer* component in order to generate a different type of contextual element. For example, combining a location context element with a time context element in order to provide a tide-level context element. A synthesizer

could also use one or more sensors generating the same type of context element in order to produce a 'best' value for that context element.

The outputs of sensor and synthesizer components are connected to a specific artefact's state via a *monitor* component. The role of the monitor component is to optimise the efficiency with which data flows from a sensor to a state. It aims to minimise the amount of data flow and processing whilst maintaining a specified quality of service threshold for a specific artefact's state. Each state has its own monitor (if it is linked to a sensor), enabling the quality of service for context element updates to be precisely specified, and hence allowing the CIS to make best use of the limited resources.
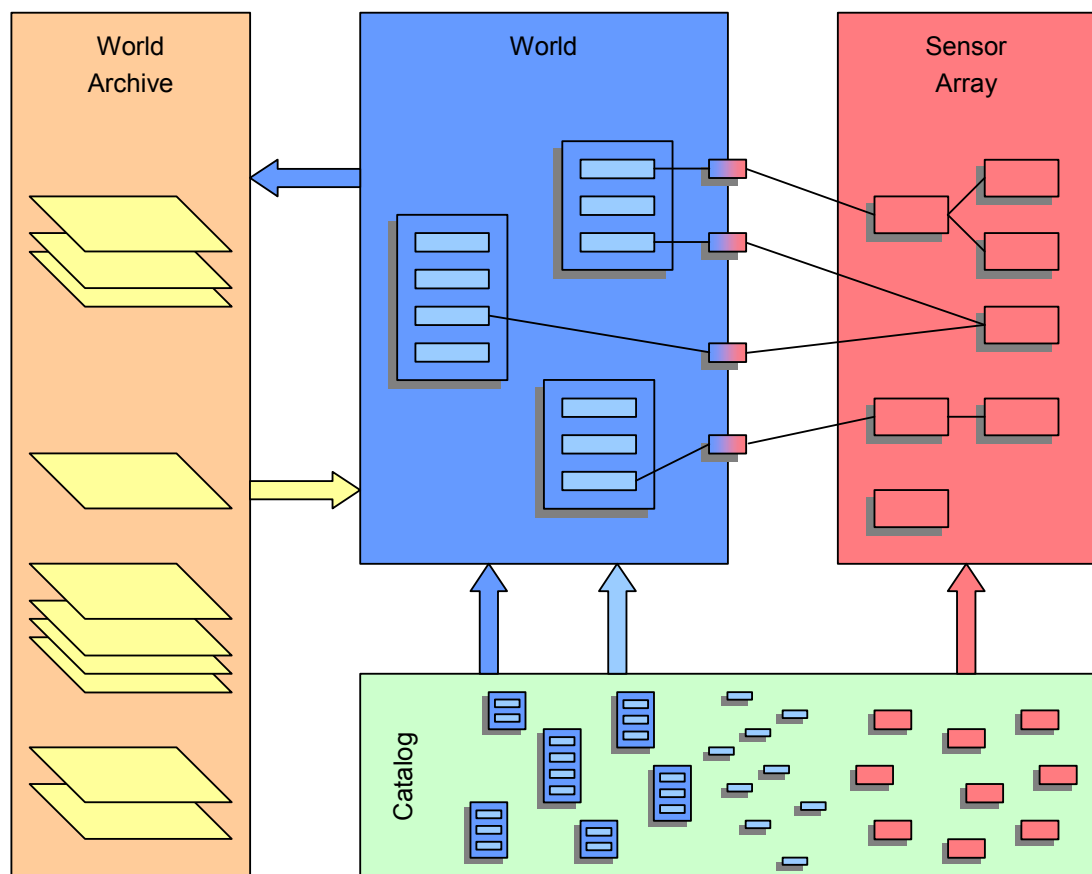
Figure 38. Service components of the CIS.

The modelling components provide the building blocks with which to construct a context model. However, to actually construct, maintain, and access the model, four service components are provided, as shown in the Figure 38.

The *catalog* service maintains a repository of reusable modelling components. Once an artefact, state, monitor, synthesizer, or sensor component has been created (e.g. a GPS receiver sensor) it is entered into the catalog from where any CIS client may use it in constructing a context model. The aim is that code written to interface with sensor devices, represent context elements, etc., is only written once and then used over and over again.

A client constructs a contextual model by taking components from the catalog service and adding them to the *world service*, which is responsible for maintaining a common context model for all clients. Within this common modelling area the clients may develop their own models or reuse existing model components inserted by other clients. For example, a new client may choose to reuse an existing artefact representing the user and her location. This is a kind of run-time reusability made possible by the CIS.

Sensor components drawn from the catalog service may be installed in the *sensor array*, which acts as a shared access medium of sensors. Rather than clients hogging a dedicated link to a sensor, and so preventing any other software from using that sensor, the sensor array enables the sharing of sensors among any number of clients. A client may utilise context element data directly from the sensor array, but more commonly a sensor will be linked to an artefact's state in the world service. The world service is the modelling space in which a client may create a representation of the parts of the world that it is interested in by inserting artefacts and states from the catalog service. The modelling space is a common area in which clients may reuse the artefacts that other clients have inserted. Some artefact's states may be updated directly by a client, but they may also be updated automatically by adding a monitor component that serves as an intelligent bridge between state and sensor. Clients may use generic monitors or develop their own custom-made monitors that use the sensors more efficiently.

The remaining service is the *world archive*. This allows clients to capture snapshots of the world service model, or a subset of it. These "snapshots in time" may be used for historic archival purposes or to save a segment of the world model so that it can be "reanimated" at a later data (e.g. like saving a document and then later opening it again).

7.2   CIS Component Design

Before we discuss the design of individual components we should establish the environment in which the CIS will be present. The CIS is designed as a device centric service. That is, a CIS should be present on every device that wishes to support context-aware software. It is not a server-based system in which client devices tap into a central context information repository. Rather each device maintains its own suite of CIS services from a first-person perspective. It is able to update its world service's context model through manual client updates or sensors that are attached or built into the device.

We have two main reasons for choosing this first-person device-centric perspective for our CIS. Firstly, although we value the concept of distributed context services, in many cases context-aware software is to be found on palmtop computer devices with no network connectivity. We therefore need a service that is capable of running in this disconnected environment. Secondly, we have found that context information from the immediate environment of a device is generally of the most value to the context-aware software that it supports, e.g. the location of the user. This is quite clear when we consider that context-aware software tends to be focused on adapting and reacting to the physical environment of the user, i.e. the immediate environment of the device.

We envision every context-aware device having its own view on the world through its own CIS. However, just as people share information about their environment it can also be useful for these devices to share information about their perceptions of the world. For example, if the device has no location sensor then it would be useful if it could connect to a neighbouring device's CIS and access its location information. We describe a method of doing just that in the next chapter. But first we discuss in more detail the design of CIS components in a stand-alone environment.

7.2.1   Sensors

As is the case for all of the various modelling components, sensors can be realised as software components, e.g. Java objects. The CIS has been designed from an open architecture perspective where we would like to encourage developers to create their own components. Considering the plethora of sensor systems on the market this would

seem the only feasible approach in providing a CIS that aims to offer extensive context element support. Although we typically talk about CIS sensor objects mapping to physical sensor devices they can just as validly gather context data directly from other software or data sources (e.g. looking at the user's electronic diary to find out when he is in a meeting). The function of the object is still that of a sensor irrespective of the source of the resulting state and the method of its retrieval.

To allow third party developers to create and add their own sensors to a CIS we need to establish a common interface through which other CIS components and clients may interact with the sensor. The following functions define this interface:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc.

➢ `Configurator GetConfigurator():` returns a standard configuration object through which the sensor parameters may be queried and adjusted in a sensor-independent manner.

➢ `Status GetStatus():` returns the current status of the sensor (e.g. error conditions).

➢ `DateTime GetLastUpdated():` returns the date/time the sensor was last updated.

➢ `Boolean IsOnline():` checks if the sensor is online.

➢ `Boolean SetOnline():` sets the sensor to be online and operational.

➢ `Boolean SetOffline():` sets the sensor offline (i.e. the software module suspends its interaction with the physical sensor).

➢ `Descriptor[] GetReturnStates():` returns a list of Descriptor objects describing the CIS states (i.e. context elements) that the sensor is able to supply.

➢ `State GetState(Descriptor):` returns the current value of the specified CIS state.

This common interface enables sensors to be operated and used through a generic mechanism, and hence new sensors can be transparently added to the CIS. Nevertheless there will almost certainly be some sensor specific configuration necessary when it is instantiated in the sensor array: if for example, specifying the serial port that the device is connected to, the baud rate it is operating at, etc. For this initial configuration process the CIS may require the user to specify the values of some parameters. However, with the information contained in the sensor's configurator object the CIS can dynamically construct a user interface and supply some sensible default values. Once the sensor is installed in the sensor array it may be discovered, manipulated, and queried by other CIS components and clients. Note that the CIS does not prevent the sensor offering its own supplementary proprietary interface in addition to the standard CIS interface.

7.2.2   Synthesizers

Synthesizers output a state that is a function of one of more state inputs from other sensors and/or synthesizers. Their function is to convert, abstract, combine, or otherwise process the input data to generate a new state. The internals of the synthesizer object therefore focus on processing of state, unlike the sensor objects, which focus on retrieving state from an external source. However, at an interface level synthesizers are the same as sensors bar the additional requirement of one or more state input streams. The interface to a synthesizer is therefore a superset of a sensor. The additional functions are listed here:

➢ `SourceName[] GetSourceNames()`: returns a list of names/identifiers for the sensor sources that it requires.

➢ `Descriptor GetSourceType(source_name)`: returns a descriptor object of the state required for the specified source.

➢ `SetSource(source_name, SensorID)`: specifies one of the source sensors.

➢ `SetSource(source_name, ArtefactID, state_name)`: specifies an artefact's state to be used as a source.

➢ `SetUpdateMode(mode):` defines the strategy for generating the synthesizer's output state. It may be evaluated on request or continually updated at pre-specified intervals.

➢ `UpdateMode GetUpdateMode():` returns the current update mode.

➢ `SetUpdateInterval(time):` specifies the update frequency for the timer-based update mode.

➢ `Time GetUpdateInterval():` returns the current update interval.

➢ `Update():`explicitly requests the synthesizer to generate a new state value.

Note that an artefact state may also be specified as an input source for a synthesizer. This is useful for artefact-states where no sensor is connected and the value is being updated manually by the user or client program.

7.2.3 States

A state component represents a particular context element value. There may be more complex peer interactions between state components than for sensors or synthesizers, such as checking if one state is larger than another. However, this depends on the type of context element. For example, checking if one temperature is higher than another may be useful whereas checking if the mood of the user is larger than another is nonsensical. An additional complexity is the need to support ranges and areas of state values. To cope with these issues we have defined a class hierarchy of different general types of state. When creating a new state object the developer will pick which type of state is the most appropriate for her context element. In most cases the choice should be intuitive.

Figure 39 only shows the state-unit branch of the hierarchy of state classes (shaded boxes indicate instanciable classes). State unit classes represent a single context element value, whereas the state containers handle ranges and areas of values. We do not consider this tree as complete; the key root classes may need to be expanded upon as context elements are discovered that do not fit with the existing structure. However, we

believe that this is a good starting point that will accommodate a large number of context elements.
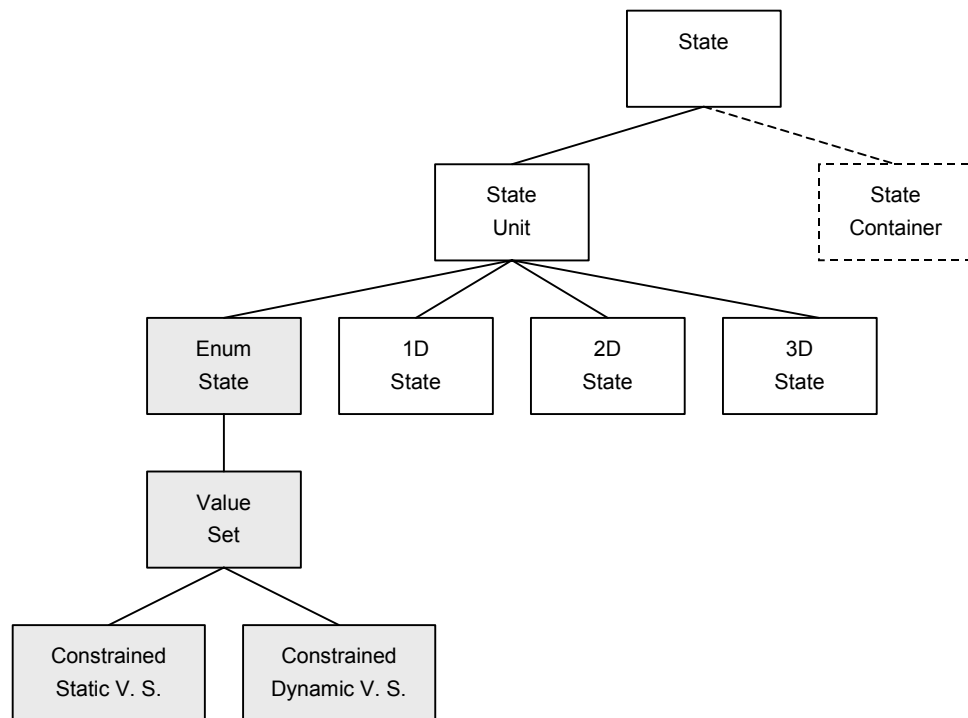
Figure 39. The StateUnit branch of the State class hierarchy.

The `State` class is the root of both unit and container types of state and has the following interface:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc.

➢ `Configurator GetConfigurator():` returns a standard configuration object through which the state's parameters may be queried and adjusted in a state-independent manner. A parameter, for example, may be the datum being used in a latitude and longitude type.

➢ `String ToString():` returns a string representation of the current state value.

➢ `Boolean Equals(state):` checks if the supplied state is equal to this one.

➢ `State Duplicate():` returns a duplicate copy, or clone, of the state.

The purpose of the next class in the hierarchy, the `StateUnit` class, is purely for maintaining a clear class structure, it adds nothing to the functional interface of the `State` class.

At the next hierarchy level we classify state units into four possible types: enumerated values (e.g. color), one-dimensional values (e.g. temperature), two-dimensional values (e.g. x,y location), and three-dimensional values (e.g. location with altitude). We have found that most states we have wanted to model will fit into this classification scheme. However, if a state should not adequately fit into any of these categories then the developer can simply derive their state from the generic `StateUnit` class. However, the state would not have the expanded interfaces of the lower classes, as described below:

1D State: contains a single value that can be compared to other values.

➢ `Boolean GreaterThan(1D_State):` checks if this state value is greater than the supplied one.

➢ `Boolean GreaterThanOrEqual(1D_State):` checks if this state value is greater or equal to the supplied one.

➢ `Boolean LessThan(1D_State):` checks if this state value is less than the supplied one.

➢ `Boolean LessThanOrEqual(1D_State):` checks if this state value is less than or equal to the supplied one.

➢ `Translate(1D_State):` translates the current state value by the specified amount.

2D State: contains two 1D States for the x and y dimensions.

➢ `SetX(1D_State):` sets the x state.

➢ `SetY(1D_State):` sets the y state.

➢ `1D_State GetX():` gets the x state.

➢ `1D_State GetY():` gets the y state.

➢ `Translate(1D_State, 1D_State)`: translates the current x and y states by the specified amounts.

Comparison of two-dimensional states is done by extracting and comparing their one-dimensional state values. It is often more useful to compare the individual one-dimensional states than the two-dimensional state as a whole. For example, it is useful to see if a location is west of another location or to see if it is north of another location, but probably not to see if one two-dimensional co-ordinate as a whole is greater or less than another one – the meaning is ambiguous.

3D State: contains three 1D states for x, y, and z dimensions.

➢ `SetX(1D_State)`: sets the x state.

➢ `SetY(1D_State)`: sets the y state.

➢ `SetZ(1D_State)`: sets the z state.

➢ `1D_State GetX()`: sets the x state.

➢ `1D_State GetY(1D_State)`: gets the y state.

➢ `1D_State GetZ(1D_State)`: gets the z state.

➢ `Translate(1D_State, 1D_State, 1D_State)`: translates the current x, y and z states by the specified amount.

As with two-dimensional states the values are compared by extracting and comparing the one-dimensional states.

Enumerated State: contains a string-based value that can represent non-dimensional abstract states such as giraffe behaviour (e.g. feeding, running, defecating, ruminating). Any string based value can be set with the following interface:

➢ `Set(String)`: sets the value to the given string.

➢ `String Get()`: returns the current string value of the state.

Value Set Enumerated State: contains the same string values as the Enum State but creates a set of usable values by observing the data entered with the `set()` function. The two derived classes: Constrained Static Value Set and Constrained Dynamic Value Set, provide exactly the same interface but with different semantics. The Constrained Static Value Set only allows the values predefined in the value set to be entered with the `set()` function. The Value Set values are set at start-up and are then immutable. The Constrained Dynamic Value Set offers the same functionality but allows explicit modification of the value set during runtime. The common interface for all three sets is shown here:

➢ `AddToValueSet(String):` adds the specified string to the value set.

➢ `RemoveFromValueSet(String):` removes the specified string from the value set.

➢ `Boolean IsInValueSet(String):` checks if the specified value is contained within the value set.

➢ `String[] GetValues():` returns an array of all the values.

Using the enumerated states a client can define context elements based on string values, and choose from a number of ways to enforce (or not enforce) the set of possible values.
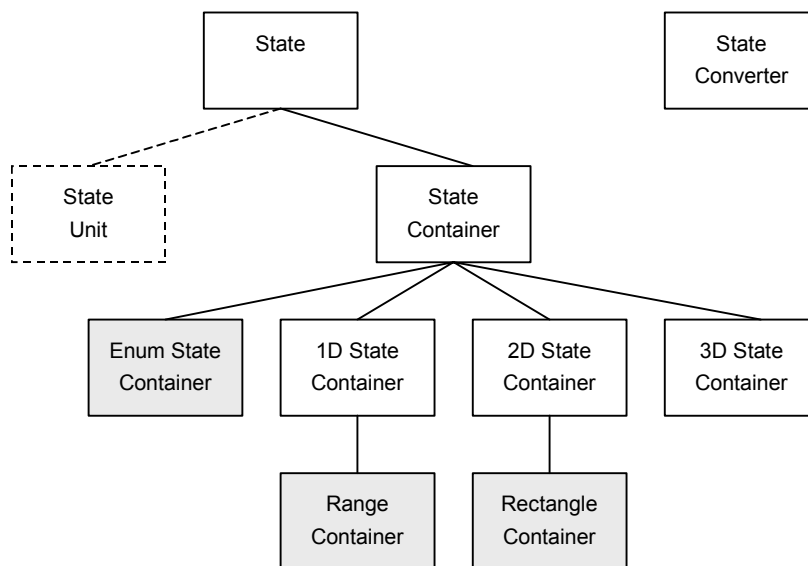
Figure 40. The StateContainer branch of the State class hierarchy.

This concludes the support for state units. Figure 40 illustrates the right-hand branch of the State class structure that offers the container support.

As previously mentioned, container classes allow the representation of ranges, areas, or volumes of state rather than the singular values of the state unit classes. However, containers are defined by a number of state units (e.g. a range container is defined by two 1D state unite objects) and will often be used in conjunction with them (e.g. checking if a point lies within a range).

Similarly to the `StateUnit` class, the `StateContainer` is an abstract class that simply defines the general interface for state containers:

➢ `Boolean Contains(StateUnit):` checks if the specified state unit lies within the container.

➢ `Boolean Contains(StateContainer):` checks if the specified state container lies within the container.

➢ `Boolean Overlaps(StateContainer):` checks if there is an overlap of values between this container and the supplied one.

Enumerated State Container: defines a container as a number of enumerated values, i.e. a container of enumerated states is a set.

➢ `Clear():` empties the container of all enum states.

➢ `Add(EnumState):` adds enumerated state unit to the container.

➢ `Remove(EnumState):` removes the specified enumerated state unit from the container.

➢ `Integer Size():` returns the size of the container, i.e. the number of state units.

➢ `State Get(number):` returns the specified state unit.

1D, 2D, and 3D State Containers: these are little more than place-holders that maintain the clarity of the class structure. They add no additional functions to the `StateContainer` interface.

Range Container: this is one type of one-dimensional container. It defines a value range in terms of a two one-dimensional states.

➢ `SetStart(1D_State)`: sets the one-dimensional value that defines the start of the range.

➢ `1D_State GetStart()`: returns the start of the range.

➢ `SetEnd(1D_State)`: sets the one-dimensional value that defines the end of the range.

➢ `1D_State GetEnd()`: returns the end of the range.

Rectangle Container: this is one type of two-dimensional container. It defines a rectangular value area with two two-dimensional states.

➢ `SetBottomLeft(2D_State)`: sets the two-dimensional value that defines the bottom-left corner of the value area.

➢ `2D_State GetBottomLeft()`: returns the bottom-left value.

➢ `SetTopRight(2D_State)`: sets the two-dimensional value that defines the top-right corner of the value area.

➢ `2D_State GetTopRight()`: returns the top-right value.

This concludes the basic set of classes from which states can be defined. In the two class diagrams the shaded boxes indicate the classes that may be instantiated with no further development effort (namely the enumerated states, the enumerated state container, the range container, and the rectangle container). In an implementation of this CIS it is likely that a number of other common states would also be added for convenience, e.g. location, time, speed, temperature, etc. Certainly, there are many more types of state container that will be required, e.g. a polygon representation of a 2D container. However, third party developers may freely extend the state hierarchy including, of course, more application-specific state support.

We wish to encourage third parties to develop their own state components but we would also like to promote interoperability between different components. In order to

do this we introduce the `StateConverter` component that has the simple remit of converting one state to another. The abstract interface for the state converter component is given below:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc.

➢ `DescriptorList GetInputStates():` returns a list of descriptors for the states that it can convert from. Note that conversions are directional, i.e. support for the conversion from one state to another does not guarantee that the reverse translation holds true.

➢ `DescriptorList GetOutputStates():` returns a list of descriptors for the states that it can convert to.

➢ `Boolean CanConvert(Descriptor Input, Descriptor Output):` enables a client to check if a conversion between the two states is possible.

➢ `State GetConversionErrorMargin(Descriptor Input, Descriptor Output):` returns the error margin for the conversion process.

➢ `State Convert(State input, Descriptor OutputType):` accepts the input state and returns the state value converted to the state described by `OutputType`.

There are no state converters supplied with the CIS but as third parties create new types of state they may also develop converters to ensure compatibility with existing components. The converters will be used either directly by clients, or indirectly by monitors that may automatically select one from the catalog in order to convert a sensor's output to a form compatible with an artefact state.

7.2.4 Artefacts

An artefact is of a slightly different nature to the other components that we have discussed thus far, in terms of the way in which the CIS can be extended. New types of state, synthesizer, or sensor components are created through writing new program code for objects that inherit the interface and functionality from one of the CIS base classes.

However, the single artefact class of the CIS is used to represent any type of artefact: person, computer, giraffe, etc.

An artefact component simply consists of a name and a set of states that describe a specific real or imaginary object, i.e. the artefact is not just a random aggregation of states, rather they are all properties of the same object. If a CIS client would like to create, for instance, a representation of a user and their location, then it would simply instantiate a new artefact, name it appropriately and add a location state. To further minimise the work of third parties to extend the CIS, artefact instances may be added to the CIS catalog as *artefact templates* from which other artefacts may be created. For example, a developer may create an artefact presenting a kongoni and add that to the catalog as a template so that all that is required to create a kongoni in the future is to pick the artefact template from the catalog.

We would like to offer the same simplicity in creating and adding new types of sensors, synthesizers, and states. Unfortunately this is not possible as new types of these components require additional processing code not, as is the case with artefacts, just new data elements.

The artefact component is utilised through the following interface:

➢ `Descriptor GetDescriptor():` in addition to storing the version information, author, etc., that all components store in their descriptors, the artefact descriptor also specifies the template name used to construct this artefact (if any) and whether this artefact itself is to be regarded as a template.

➢ `String ToString():` returns a string representation of the artefact, i.e. its name and state values.

➢ `Boolean Equals(Artefact):` checks if the supplied artefact is equal, state by state, to this one.

➢ `Artefact Duplicate():` returns a duplicate copy of the artefact.

➢ `String GetName():` returns the name of the artefact.

➢ `SetName(String):` sets the name of the artefact to the specified string.

➢ `String GetTemplateName():` returns the name of the template used to construct this artefact (if one was used and this is not a template itself).

➢ `String[] GetStateNames():` returns a list of names for the states held within the artefact.

➢ `State GetState(String state_name):` used to access a named state.

➢ `SetState(String state_name, State):` used to update the value of a named state.

➢ `AddState(String state_name, State):` adds the state to the artefact under the specified name.

➢ `RemoveState(String state_name):` removes the state of the specified name from the artefact.

➢ `AddMonitorWatcher(Monitor, String state_name):` allows a monitor component to watch for accesses to a particular state (see next section on monitors for more information).

➢ `RemoveMonitorWatcher(Monitor, String state_name):` allows a monitor's interest in watching a particular state to be removed. If no state is specified then all interests for that monitor are removed.

If a template was used to construct an artefact then the artefact remembers this name even if the states it contains are changed. This information can later be used to hone artefact searches in the world service, i.e. searching by name and type rather than just name. Note that the creation of artefacts from templates, and the creation of templates themselves, is handled by the catalog service.

Currently templates only exist as entities unto themselves but in future work we would like to investigate the possibility of providing a hierarchy mechanism in which families of templates can be created, similar in principle to inheritance used in object oriented programming languages.

7.2.5 Monitors

Monitors connect sensors to artefact states and regulate, and attempt to optimise, the flow of data between the two. They are more similar in construction to sensors than artefacts, as new monitor components are created by programming new classes. However, developing new monitors is not always necessary as they serve only to improve CIS performance (by optimising sensor-to-state data flow) rather than to add new functionality. Third party developers may use one of the generic monitors shown in the class diagram below.
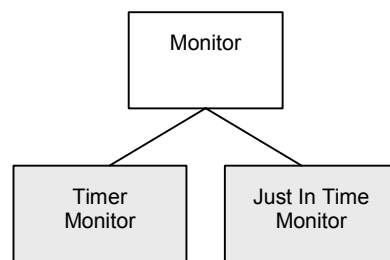
Figure 41. Monitor component class hierarchy.

The Timer Monitor performs an update of the artefact-state it is connected to at specified time intervals, whereas the Just In Time Monitor updates the artefact-state whenever the state value is accessed from the artefact. The interface to the generic monitor base class is given below:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc.

➢ `SetArtefactState(ArtefactID, String state_name):` connects the monitor to the specified artefact's state. (Can also be used to set the artefact-state to none).

➢ `SetSensor(Sensor):` connects the monitor to the specified sensor. (Can also be used to set the sensor to none).

➢ `SetArtefactStateAsSource(ArtefactID, String state_name):` allows the source to be an artefact state. (Can also be used to set the artefact-state to none).

➢ `Update():` explicitly requests that the monitor updates the artefact's state.

➢ `DateTime GetLastUpdated():` returns the time that the monitor last updated the state.

➢ `SetOnline():` turns the monitor on.

➢ `SetOffline():` turns the monitor off.

➢ `Boolean IsOnline():` checks if the monitor is online.

The Timer Monitor adds the following functions to the interface:

➢ `SetUpdateInterval(Time):` specifies the interval between state updates.

➢ `Time GetUpdateInterval():` returns the interval between state updates.

The Just In Time Monitor adds one function to the interface:

➢ `StateValueAccessed():` this function may be called by the artefact when an access attempt is made on the state whose value the monitor is supplying.

This callback function of the Just In Time Monitor allows the monitor to supply a new value to the artefact just before the artefact gives the value to a client requester. The monitor registers its interest in a state with the artefact that owns it so that this notification may be made.

Monitors conclude our description of the prototype CIS modelling components, we now continue with the service components. Unlike many of the model components, the service components are not intended to be extended by third parties. The four core services remain the same across different platforms, providing a reliably consistent interface to the CIS.

7.2.6 Catalog Service

The catalog manages the CIS's repository of modelling components. Clients may search or browse the catalog for particular components or particular types of component. They may also add their own components to the catalog. It is this ability to discover new components and to extend the range of modelling components that is the catalog's key objective. Without the catalog acting as a central manager for the CIS's modelling

components it would be difficult for clients to add, distribute, or discover modelling components outside of the core set provided with the CIS.

The catalog is accessed through the following interface:

➤ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc. of the catalog.

➤ `AddComponent(ModelComponent, Descriptor):` adds any type of modelling component to the catalog. The service can then use the Descriptor information to store the component in the appropriate part of the catalog and to resolve conflicts such as replacing an old version of a component with a new version.

➤ `RemoveComponent(Descriptor):` removes the component specified by the descriptor, but only if the component is not currently in use.

➤ `Boolean ComponentExists(Descriptor):` checks if the specified component exists within the catalog.

➤ `DescriptorList FindComponents(DescriptorSearch):` the `DescriptorSearch` object allows a client to express search queries for any of the items of information stored within a `Descriptor` object. For example, this function could be used to find all components added by a certain author.

➤ `Integer HowManyComponents(ComponentType):` checks how many components of the given type are stored in the catalog, e.g. how many sensor components.

➤ `DescriptorList GetComponentDescriptors(ComponentType):` retrieves a list of Descriptors for all the components of the given type within the catalog.

➤ `Descriptor GetComponentDescriptor(ComponentType, Integer):` retrieves the Descriptor of the single component specified.

➤ `ModelComponent GenerateComponent(Descriptor):` generates an instance of the `ModelComponent` specified by the descriptor.

What we have not addressed in the catalog, and all the other service components, is security and access rights to these functions. Obviously some control is required to prevent client programs from maliciously removing or corrupting components within these services. However, for the purposes of constructing a basic CIS design we will assume that all programs are trustworthy.

In future versions we would also like to add more sophisticated search functions for specific types of component, e.g. to find all sensors that output a location state.

7.2.7 Sensor Array Service

The sensor array provides a common area that can be populated with active sensors. The objective of the sensor array is to allow runtime reusability of sensors, e.g. for any number of clients to share access to the same location sensor without conflict. Normally clients will work with sensor data indirectly by extracting values from artefact's states in the world service. However, a limited degree of interaction is required with the sensor array to install and register interest in sensors using the following interface.

➤ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc. of the sensor array.

➤ `SensorID AddSensor(Sensor, Time RemovalTimeout):` adds a sensor to the array (hence making it accessible to all other clients) and returns its unique identifier within the sensor array. The sensor array may remove the sensor if it has not been used for the `RemovalTimeout` period.

➤ `Integer HowManySensors():` returns the number of sensors in the sensor array.

➤ `SensorIDList GetAllSensorIDs():` returns a list of IDs for sensors within the sensor array.

➤ `SensorIDList  FindSensorIDs(Descriptor  StateType,  Boolean ExactStateMatch):` searches for any sensors that can supply the specified type of state, e.g. finding any sensor that can supply location information. If

`ExactStateMatch` is specified then only an exact match will suffice, even if a sensor supplying a compatible/convertible type of state is found.

➢ `Descriptor GetSensorInfo(SensorID):` returns the descriptor for a particular sensor.

➢ `Sensor CheckoutSensor(SensorID, Function Callback):` returns the specified sensor. The `Callback()` function is used by the sensor array to check if the client who checked out the sensor is still alive and interested in the sensor, and also to communicate that a sensor is about to be removed.

➢ `ReturnSensor(SensorID):` allows the client to de-register its interest in the sensor.

➢ `Integer NoOfClients(SensorID):` returns the number of clients who currently have the sensor checked out.

➢ `ConnectMonitor(SensorID, Monitor):` connects the supplied monitor to the sensor.

➢ `DisconnectMonitor(SensorID, Monitor):` disconnects the monitor from the sensor.

➢ `Integer NoOfMonitors(SensorID):` returns the number of monitors that are currently connected to the specified sensor.

➢ `RemoveSensor(SensorID):` forces the removal of a sensor from the sensor array.

The CIS lets any client enter a sensor into the sensor array. Clients may then "check out" a sensor if they are interested in using it (including the client who originally added the sensor to the array). When a client is finished with the sensor it should return it to the array, although many clients may check out the same sensor simultaneously. The sensor array maintains a registry of clients who have a sensor checked out. When there are no longer any clients using a sensor, and after the given removal timeout period has expired, the sensor will be automatically removed from the sensor array. To prevent clients who fail to de-register from permanently "locking" a sensor the `Callback()` function may be used to periodically check if clients are truly interested in the sensor. If

a client fails to respond then they will be considered to have returned the sensor. Normally all sensor removal will be automatic, but a forced removal of a sensor is possible using the `RemoveSensor` function. In such a scenario any clients that currently have the sensor checked out will be notified of the removal via their callback function.

The automatic management and removal of sensor resources means that clients have little work to do other than the original instantiation of the sensor in the array. The originator client, and any other client, may then use the sensor and not worry about its disposal, which will occur automatically when there is no more interest in the sensor.

7.2.8 World Service

The world service provides a context model of the real and imaginary world in the form of a set of artefacts, representing objects, and their states, representing context elements. It is at the heart of the CIS, providing client applications with a view onto the world around them, i.e. it is the means with which clients obtain their context-awareness. Similar in principle to the sensor array, the world service aims to offer a common medium through which a context model can be constructed and shared by many clients. The key activities are adding new artefacts and finding the value of a state of an artefact that already exists. The following interface facilitates such activities:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc. of the world service.

➢ `ArtefactID AddArtefact(Artefact, Time RemovalTimeout):` adds an artefact to the world, hence making it accessible to all other clients, and returns a unique identifier by which it can be referred to. Like the sensor array, the world service may remove the artefact if no client has been interested for the `RemovalTimeout` period.

➢ `Integer HowManyArtefacts():` returns the number of artefacts in the world service.

➢ `ArtefactIDList GetAllArtefactIDs():` returns a list with the identifier of every artefact in the world service.

➢ `DescriptorList GetAllArtefactTemplates()`: returns a descriptor list of all the artefact templates currently in use in the world.

➢ `ArtefactIDList Search(String Name, Descriptor Template, Descriptor StateType, StateContainer, Boolean ExactStateMatch, ArtefactIDList Scope)`: searches for an artefact using any combination of the supplied search parameters, i.e. the client need only specify the parameters that they are interested in using for a search. The `Name` parameter allows the client to specify a search string that will be compared to the artefact's name. The `Template` parameter allows the client to specify the template from which the artefact was created (e.g. to find people artefacts). The client may also specify that the artefact must have a particular type of state, and that this state must lie within the given `StateContainer` region. If the client exerts an `ExactStateMatch` then only states that exactly match the `StateType` descriptor are included in the search. If it is not set then other state types that are not exactly the same, but that are compatible with this type of state, are also included. If the client is only interested in a particular subset of the world then they can specify this in the `Scope` parameter. This helps to improve system performance by limiting the scope of the search. It also can be used in an iterative search process where the results from one search can be fed into the subsequent search, so gradually honing down the list of artefacts.

➢ `SearchID StandingSearch(String Name, Descriptor Template, Descriptor StateType, StateContainer, Boolean ExactStateMatch, ArtefactIDList Scope, Function Callback)`: this search function has the same search semantics as the previous function but it is kept as a standing search in the CIS. The client is continually informed through the supplied `Callback()` function of artefacts that now match and now no longer match as a result of the world changing.

➢ `ArtefactIDList CurrentArtefactList(SearchID)`: returns the list of artefacts that currently fall within the parameters of the specified standing search. This may be used by the client if for some reason it loses track of the individual add and remove commands issued by the CIS through the `Callback()` function.

➢ `RemoveSearch(SearchID)`: removes the specified standing search from the CIS.

➤ `WatchID AddWatch(ArtefactID, String StateName, Time MaxUpdateFrequency, Function Callback)`: registers an interest in a particular artefact state (or whole artefact if no state is specified). If the artefact/state changes then the client is notified of the new state value through the `Callback()` function. The `MaxUpdateFrequency` may be used to regulate how often the CIS informs the client of a state change, so preventing saturation of the client with update notifications.

➤ `RemoveWatch(WatchID)`: removes the specified watch from the CIS.

➤ `Artefact CheckoutArtefact(ArtefactID, Function Callback)`: returns the specified artefact. The `Callback()` function is used by the world service to check if the client who checked out the artefact is still alive and interested in the artefact, and also to communicate that an artefact is about to be removed.

➤ `ReturnArtefact(ArtefactID)`: allows the client to de-register its interest in the artefact.

➤ `State GetArtefactState(ArtefactID, String StateName)`: returns the current state value of the specified artefact state. This function can be used by clients to quickly ascertain the value of an artefact state without having to go through the process of checking the artefact out, examining the state, and returning the artefact.

➤ `Descriptor GetArtefactInfo(ArtefactID)`: returns the descriptor for the specified artefact.

➤ `Integer NoOfClients(ArtefactID)`: returns the number of clients who currently have the artefact checked out.

➤ `ConnectMonitor(ArtefactID, String StateName, Monitor)`: connects the supplied monitor to the specified artefact state.

➤ `DisconnectMonitor(ArtefactID, String StateName, Monitor)`: disconnects the monitor from the state.

➢ `Integer NoOfMonitors(ArtefactID)`: returns the number of monitors that are currently connected to the specified artefact.

➢ `RemoveArtefact(ArtefactID)`: forces the removal of an artefact from the world service. Clients using the artefact are notified of its removal.

The world service provides comprehensive capabilities for observing context changes. Clients may view these context changes from the perspective of a particular artefact, watching how its individual states change, or from the perspective of context region, observing how artefacts enter and exit the region as their states change. Additionally, the CIS allows the client to access artefacts and state information through either polling or event delivery mechanisms, i.e. performing one-off queries/searches or registering continuous watches/searches.

As with the sensor array component the world service is self-maintaining, automatically removing artefacts that no longer have any interested clients.

7.2.9 World Archive Service

Context-aware applications may not just require access to the current context, but also to past contexts. For example, memory prosthesis [Lamming, Brown 1994 - #76] applications can help the user find files and other resources through specifying the context in which they were last worked with. The world archive allows the client to save and store a *slice* of the current CIS. These slices may be used for archival purposes, providing a kind of context memory.

In computer systems persistence between sessions is common, so that an application retains the same state and appearance the next time it is opened. The world archive slices may be used for this purpose, allowing the *reanimation* of an archived CIS slice in the future. For example, rather than having to execute the same set of operations to create a context model each time a client program starts, it can instead save the slice of the CIS it is interested in when it shuts down and then reanimate the slice when it restarts.

The interface the world archive service provides to carry out such operations is as follows:

➢ `Descriptor GetDescriptor():` returns a descriptor object containing version information, author, etc. of the sensor array.

➢ `CreateSlice(String SliceName, ArtefactList, Boolean SaveLinks):` saves the subset of the world specified by the artefact list in a new slice that will be named according to `SliceName`. If the client specifies to `SaveLinks` then descriptors of the monitors and sensors that each artefact is connected to are also filed. Storing the links enables the CIS to not only reanimate the artefact but also the connections it had with the sensor system.

➢ `DescriptorList ReanimateSlice(String SliceName):` reanimates the slice into the real world. If the slice contains an artefact that already exists in the world then this artefact can be used by the client instead of reanimating the one stored in the slice. The same applies in the reanimation of sensors and monitors. The archive service returns a list of the descriptors for objects that were not reanimated.

➢ `Integer HowManySlices(DateTime start, DateTime):` returns the number of slices that were archived within the specified time range.

➢ `SliceNameList GetSliceNames(DateTime start, DateTime):` returns the names of slices that were archived within the specified time range.

➢ `Integer HowManyArtefacts(String SliceName):` returns the number of artefacts in the given slice.

➢ `ArtefactIDList GetAllArtefactIDs(String SliceName):` returns a list with the identifier of every artefact within the given slice.

➢ `DescriptorList GetAllArtefactTemplates(String SliceName):` returns a descriptor list of all the artefact's templates contained in the slice.

➢ `ArtefactIDList Search(String SliceName, String Name, Descriptor Template, Descriptor StateType, StateContainer, Boolean ExactStateMatch, ArtefactIDList Scope):` searches the slice for artefacts matching the given parameters. Operates in the same way as the `Search` function in the world service.

➢ `State GetArtefactState(String SliceName, ArtefactID, String StateName)`: returns the value of an artefact-state within the specified slice.

➢ `Descriptor GetArtefactInfo(String SliceName, ArtefactID):` returns the descriptor for the specified artefact.

➢ `RemoveSlice(String SliceName)`: removes the slice of the specified name.

Unlike the world service, which allows clients to checkout artefacts and operate on them directly, the world archive service maintains tight control over the contents of each slice. Access to the artefacts and states therein must be carried out through the read-only interface, preventing clients from "altering history". One could argue that history may need to be corrected, e.g. if a sensor was malfunctioning and producing spurious data. However, those spurious results could also be useful, e.g. in tracking down problems with sensors, so we prefer to keep the archive authentic to the actual data recorded.

7.3   Evaluating the Prototype Design

It is not possible to exhaustively evaluate the prototype design until an implementation has been fully developed and thoroughly tested with a number of different types of client application (as mentioned earlier, an implementation of the service components has yet to be done). However, we can make a preliminary evaluation of the design alone based on the criteria that we used in the previous chapter for comparing the work of various CIS projects. That chapter commenced with a discussion of the nature of context and the nature of context-aware applications, and then went on to examine in detail the possibilities for a CIS within the context of a five-layer model. In this section we shall assess our CIS prototype in terms of each of these aspects.

7.3.1   The Nature of Context and Context-Aware Applications

The first thing we noted about the nature of context was that there was not always a clear distinction between context and content. Our prototype CIS design simply represents sets of states that may or may not be dynamically updated from sensors. It leaves it to the application to determine a particular context or content viewpoint of the data. The CIS tries to manage the complexity of context by separating states from

sensors and by providing an environment in which new components can be added, discovered and operated in a standard fashion. The state and sensor components also seek to hide the complexity of context acquisition, conversion, and representation, beneath a simple client interface.

The CIS has no bias to any particular type of context element. New states or sensors for any type of context element can be added to the system. The CIS also supports the concept of context elements as attributes, where each state is embedded in an artefact representing a real or virtual object. To support the potentially more complex states of virtual objects specialised state components may be developed, e.g. a logical expression state that combines a number of other states with logical operators.

In the section on the nature of context-aware applications we first mentioned that there are many different classes of context-aware application. The CIS provides context provision support at a basic enough level to be application neutral but to still provide a number of useful facilities that make accessing context information a much easier process for client applications. The resource hogging nature of context-aware applications is resolved by devolving responsibility for most context-aware operations to common CIS services that can be optimised and shared by any number of CIS clients. This delegation of context-aware duties to the CIS also dramatically reduces the development costs of imbuing an application with context-awareness.

The issue of diverse computing environments is an implementation problem rather than something that will affect our design. However, the CIS is certainly well suited to be implemented in a platform independent language such as Java.

The final comment on the nature of context-aware applications was that the greater the support of context the greater the sophistication of the application behaviour. The CIS design seeks to enable the extension of context support through the ability to develop and add new state, sensor, and artefact components. It also aims to promote reuse of these components through a set of shared services through which a client may (a) discover and use any of the existing components, and (b) also explore and add to the common context model shared among clients.

7.3.2   Five Layer CIS Support

After addressing the nature of context-aware applications the previous chapter proposed that developers wishing to create such applications currently face a situation similar to that of developing an application's GUI without any supporting libraries or a common windowing environment. The CIS design addresses this problem by offering a core context service to gather, model, and provide context information. In the previous chapter we also examined a number of services in the context of a five layer support model, and we will now do the same for our prototype CIS design.

> **Layer 1: Sensor Support.** We divided sensor support into fixed and extensible categories. The prototype CIS design offers extensible support at runtime, allowing sensors to be added to the catalog and sensor array services without needing to suspend the operation of the system. Clients can discover and use new sensors as and when they are added to the sensor array. That is, the prototype design offers the highest level of sensor extensibility.

> **Layer 2: Context Data Extraction.** The CIS supports the full range of context extraction mechanisms. Values can be transferred from a sensor to an artefact's state via a monitor with no further processing. However, abstraction of context values and sensor chaining can also be performed using synthesizer components, allowing the context to be processed, transformed or fused with other data whilst en route to an artefact's state. Both client driven and model driven quality of service is supported in the CIS design, with monitors used to optimise the flow of data between sensors and artefact states. Each artefact's state has its own monitor so that the flow of data can be customised to suit the characteristics of the object whose state is being updated. Rather than using the generic monitors provided, third parties may develop their own monitors in order to optimise sensor interaction based on high-level application semantics that would not be possible for the CIS to foresee. Monitors also allow referencing (the ability to supply one artefact-state value from another artefact-state value) by accepting an artefact-state as a source instead of a sensor or synthesizer.

> **Layer 3: Context Data Type Support.** Being application agnostic, the CIS design supports a diverse set of context elements (i.e. states) and allows this support to be

extended at runtime with the addition of new components to the catalog service. As soon as the components are added to the catalog they may be introduced and used in the world service's context model.

➢ **Layer 4: Context Model Support.** An object-oriented context model is at the core of the CIS design. The world service presents context information in the form of artefacts, representing real or virtual objects, that possess various states, representing the object's context. This model is fully extensible at runtime and supports a diffuse viewpoint. However, although the world service provides a common area in which all applications build their context model, they are not forced to contribute to a central shared model; clients may build their own diffuse models within the common area. A conscious decision was made in the design of the prototype CIS not to include any peripheral service support such as relationships, triggering, manipulators or viewers. We believe that these tend to be platform and application specific, and as such should be provided as separate service layers on top of the core CIS system.

➢ **Layer 5: Client Interface.** The CIS design supports all three communications mechanisms for providing context information. Polling is supported by explicitly executed client search commands, streaming is supported by setting "watches" on the areas of interest, and event driven delivery is supported through standing searches. Watches operate at such a low level that they effectively offer hook support too. The CIS design is oriented towards client channels but it may internally combine identical watches and standing searches, effectively supporting client-group channels. The types of delivery channels supported is more significant when dealing with a distributed CIS. The whole of the next chapter is devoted to the distribution issue and explores a number of strategies for CIS distribution.

We have attempted to address the CIS requirements suggested by other research with our initial definition of CIS requirements described in the previous chapter. However, we can take a recent study by Brown et al [Brown 2001 - #17a] as further authentication of our CIS design. The aim of their work was to outline the most compelling applications of context-aware technology. In the process they define some general requirements of infrastructure intended to capture and maintain contexts. Table 28 describes their requirements and how our CIS design fulfils them.

Requirements of Brown et al	CIS Solution
To extract context from hardware sensors like GPS receivers.	A sensor component can be developed for each physical sensor, providing the CIS with the context extracted from the physical device.
"Soft" sensors for extracting context from the user directly, or from other software or data sources.	The same sensor component can be used to model these soft sensors.
Virtual and real contexts, and switching between them, e.g. firstly representing a person's location derived from a GPS receiver and then switching to a "pretended" representation of his location.	A state of an artefact may be plugged into a sensor via a monitor, and at any time unplugged and connected to another sensor. A sensor may be reporting virtual context or the client application itself may directly insert a pretend value into an artefact's state.
Synthesis for abstraction of context and error elimination through multiple sensor inputs.	Synthesizer objects support multiple sensor/synthesizer inputs and may be used to abstract, eliminate error, or perform any other process that generates a resulting state from one or more input states.
Context memory, to record changes in context over time.	This is the function of the world archive component, which allows "snapshots" of the CIS context model to be recorded.

Table 28. Brown et al's requirements, and the CIS solutions.

To summarise, we believe that the prototype CIS design successfully addresses all of the issues and suggestions that were raised in the previous chapter and, as a result, the majority of suggestions for infrastructure from other context-aware research. We think this CIS will provide a valuable service from both the viewpoint of client applications that wish to utilise context information and the third party developers wishing to build and provide new context services.

7.4   Conclusion

In this chapter we have introduced a prototype design for a CIS that attempts to address all the issues that were raised in the previous chapter. The prototype design is based on two sets of components: a set of modelling components that provides the building blocks with which to construct a context model, and a set of service components through which clients may interact with the CIS.

Five key modelling components were proposed: sensors, synthesizers, monitors, states, and artefacts. Sensors are software modules that act as the interface between the CIS and sensor hardware/software, providing one or more current context elements. Synthesizers offer a similar context element provision service but their source data originates from one or more sensors or synthesizers whose output they abstract,

transform, or combine. At the other end of the model component spectrum are artefacts. An artefact represents a real or virtual object and the set of states pertaining to that artefact (i.e. the artefact's context). The output of a sensor or synthesizer is connected to an artefact state through a monitor component. The monitor attempts to minimise the flow of data between sensor and state whilst maintaining a specified quality of service level, so optimising the CIS performance.

The five modelling components are accessed through four service components: the catalog, sensor array, world, and world archive. The catalog is an extensible repository of reusable modelling components that any client may browse, select from, or contribute to. The sensor array provides a shared sensor medium that clients may populate with sensor components taken from the catalog. The world service provides the majority of the core functionality, allowing clients to easily construct or reuse context models and to query and/or monitor all or part of the model. The context model is the subset of the real or virtual contextual world that the CIS clients are interested in. Areas of this model that are of particular interest to a client may be stored for later reference or 'reanimation' by using the world archive service.

We concluded this chapter with a preliminary evaluation of our prototype CIS design, using the same criteria that we developed in the last chapter in our exploration of the design space of CISs and the examination of various CIS projects. Our prototype CIS design has satisfactorily addressed all of the issues that were raised in the that chapter, offering a universal context provision mechanism that is highly reusable, sharable, customisable, extensible, and resource friendly.

This chapter has only addressed CIS design in terms of a stand-alone service running on a single non-networked computer. However, the next and final chapter examines the last aspect of our work CIS work: an investigation into how CIS services may be effectively distributed.

Chapter 8:

Concepts for Distributing the CIS

The CIS that we have presented in the previous chapter is limited in scope to modelling context that can be detected by the device on which the CIS is running, e.g. through a GPS device attached to a palmtop computer that is running the CIS. This results in a rather insular view of the outside world which only provides client programs with a few pieces of their context. Those pieces focus on the host device and its immediate surroundings. However, if CISs running on different hosts could communicate with one another then any one CIS would have access to a wealth of information about the world in which it finds itself. That is precisely the aim of this chapter: to investigate how contextual information can be distributed across many CISs.

This chapter is pure theory: we did not have the time or resources to make a practical implementation and evaluation of the design concepts that we propose. It can be viewed as the start of much further work on how CISs may be distributed. The work presented is a speculative set of initial concepts, ideas and discussions for future work.

This chapter first addresses some general distribution issues for contextual information and then proposes three design concepts, which we discuss in some depth. The first design concept considers connecting CISs at a low level to facilitate CISs on separate devices in presenting a common and shared world of contextual information. The second design concept discusses distribution at a more logical level where shared worlds of distributed artifacts can be constructed in order to represent a grouping around a particular theme. The third design proposes how a set of software agents could be used to discover, manage, and dispose of *local* references to *remote* artifacts that may be of interest to a particular CIS. These three concepts form our initial thoughts on how a CIS could be distributed and, as such, are a speculation on how this field could evolve in the future.

8.1   General Distribution Issues

This section defines what it is that we would like to distribute and categorises the various communication mediums over which the distribution could take place.

8.1.1   What to Distribute

There are essentially three areas in the CIS that we can consider for distribution:

➢ **Electronic Resources.** Electronic resources are virtual entities whose context can be modelled within the CIS but whose physical presence is separate from it. For example the context of a virtual billboard may be modelled as an artifact in the CIS, but the actual billboard may be a web page stored on a remote server.

➢ **CIS Artifacts.** Some artifacts represented within a CIS on a local host may actually be of interest to a number of remote hosts's CISs. Therefore, a CIS should be capable of incorporating artifacts from remote CIS worlds into its own local world, and also making its own local artifacts available to other remote CIS worlds.

➢ **CIS Sensors.** Sensors and synthesizers are often located on the same host as is the artifact whose current context they help provide, e.g. the location of a tourist that is derived from a GPS sensor attached to her PDA. However, sometimes these sensors are remote, e.g. a weather sensor that accesses a remote meteorological station, or a vehicle tracking system that reports back to a central control centre rather than the driver.

In this chapter we concentrate on the artifact level distribution. That is, in transparently providing CIS clients with a distributed collection of artifacts that are transparently accessible through the client's local CIS world service. Therefore we are primarily concerned with the distribution of the CIS world service component (we leave other aspects of CIS distribution to future work). Additionally, rather than relying on the client to specify the sets of remote artifacts that should be incorporated into the local CIS world, wherever possible the local CIS should attempt to discover such resources on the client's behalf. This is especially important for applications such as a tourist guide, where the client may not necessarily know where to find the resources in the new

area in which it finds itself. The other two levels of distribution, electronic resources and CIS sensors, we leave for future research.

8.1.2 Communications Infrastructure

The communications mediums that the CIS may be exposed to can be categorised as:

- **Wired Network.** This is the ideal medium for creating local groups of physically remote CIS artifacts. Well-established distribution technology such as CORBA [CORBA 2001 - #23] and Java RMI [Sun 2001 - #138] can easily support the representation and manipulation of objects over these fixed wired networks. However, on the main platforms of interest to this thesis (i.e. palmtop and wearable computers) a wired link is not realistic.

- **Wireless Network.** The same well-established distribution technology may be used over wireless LANs such as 802.11. However, the necessary supporting infrastructure (base stations with wired LAN connections every 100-500 metre area) make this option prohibitive for wide areas but feasible for "hot-spot" areas such as an airport or museum.

- **Mobile & Satellite Phones.** These currently provide the only practical solution for continuous mobile communications over a wide area. It is also a low cost solution from the perspective of new physical infrastructure for each user to install, i.e. none! However, it has the negative characteristics of high cost and low bandwidth.

- **Situated Communications.** These consist of technologies ranging from infrared and Bluetooth [Bluetooth-Consortium 2001 - #11], which are relatively high bandwidth, to active badges [Want, Hopper 1992 - #145] and locusts [Kirch, Starner 1997 - #72], which are relatively low bandwidth. They provide a situated form of communications that a mobile device can exploit when being in varying degrees of proximity and alignment to the facility. The most communicatively limited technologies, such as locusts, may only be capable of transmitting small items of information from a particular place, though this could be all that is needed for some applications, e.g. a locust location beacon. The more capable technologies such as Bluetooth support a higher capacity of communications.

Depending on cost, function, and application, the CIS may need to be distributed over any one or combination of these underlying communications mediums. Therefore, the design of the CIS's distribution capabilities must take into account the varying degrees of connectivity and bandwidth, ranging from a high bandwidth and continuous connection to a low bandwidth and intermittent connection, and be able to exploit the mediums it finds available in its ever-changing environment of use. Bearing this in mind, we now proceed to presenting the first of our three CIS distribution design concepts.

8.2   The Federation Distribution Design Concept

The aim of a *federation* of CISs is to provide a client of any one CIS with transparent access to the combined resources of a group of co-operating CISs, effectively letting the client work with a group of CISs as if it were a single stand-alone CIS. There is potential for sharing all four CIS service components (the world, sensor array, catalog and world archive) in such a federation but we solely concentrate on the distribution of the world service component, i.e. the distribution of the component that provides the contextual model of the world.

A CIS federation is intended to amalgamate artifacts that are distributed across a number of remote CIS worlds into one local and location-transparent world. For example, if two CISs were to form a federation then each would subsequently present a world composed of the artifacts from both CISs. The two CIS worlds still reside on physically separate CISs but are conceptually unified and treated as one, as illustrated in Figure 42. This federation is maintained in a distributed fashion where each CIS continues in its responsibility for the artifacts that it has introduced to the world; there is no notion of any controlling CIS or hierarchy of authority. In our model, a CIS shares its whole world with other members of the federation; however, as is described later, there is a provision for privacy filters. And, as with all distributed systems that share resources, provisions must be made to overcome potential naming conflicts (but this is not addressed here).
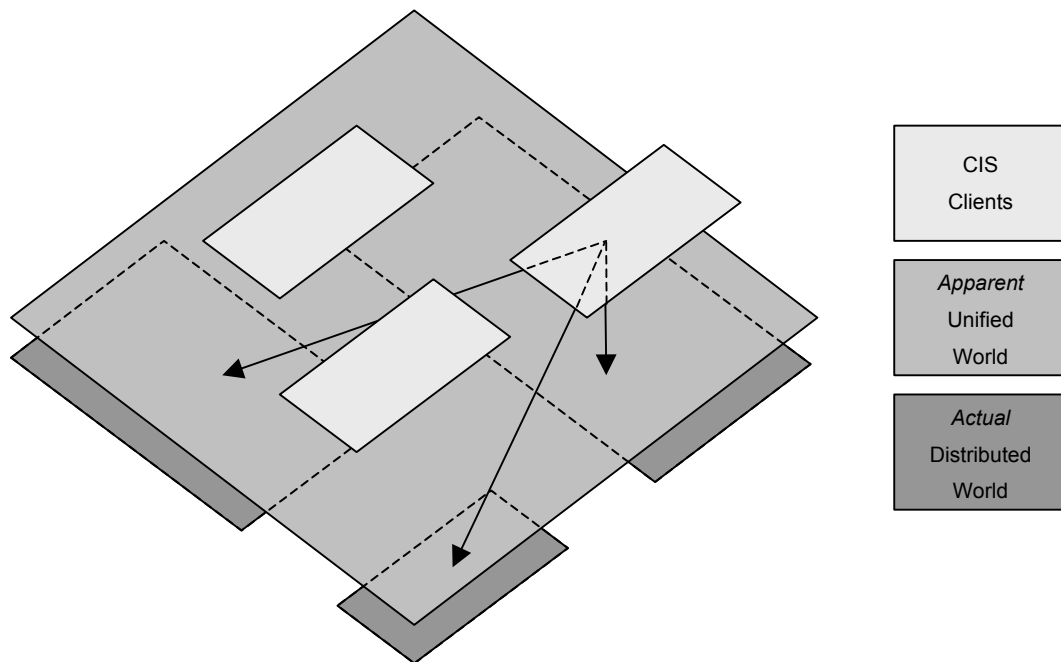
Figure 42. CIS clients transparently access a unified distributed world.

This distributed federation of equal authority members is ideal when sharing information between CISs of equal status, e.g. a group of tourists sharing their own locations with one another. Other scenarios, such as federating a tourist office's CIS and a tourist's CIS appear different but work equally well, the only difference being the imbalance in the amount of artifacts modelled by each CIS world. This imbalance is actually exactly what is required in such a situation, enabling the delegation of the bulk of the storage and modelling load to a dedicated server rather than duplicating it on lots of tourists' palmtop computers.

8.2.1   Discovering a CIS

A curious aspect to the tourist example is the lack of knowledge that a tourist has on arriving in a new location. She has no idea what CISs exist within her new environment or where to find them. However, we could exploit some situated form of communications to help her discover them, for example, locust devices hanging in an airport arrival's lounge could broadcast contact information (e.g. network addresses, phone numbers, etc.) for publicly available CISs, enabling the tourist's palmtop computer to transparently discover them. Devices such as locusts act as servers of *CIS*

keys that facilitate the connection and integration of the tourist's CIS with their new environment.

A *key server* would operate by broadcasting keys on a low-cost low-bandwidth medium, e.g. IrDA [Infrared-Data-Association 2001 - #66]. Upon discovering such a key the local CIS would contact the remote CIS via a higher-bandwidth bi-directional communications channel (the details of which would be supplied in the key). Note that more than one key server could broadcast the same CIS key, i.e. the same CIS could be used for more than one locale. Key servers would be physically planted wherever they are required, e.g. a blackbox embedded in the high street that provides a key to a CIS containing advert and public transport artifacts.
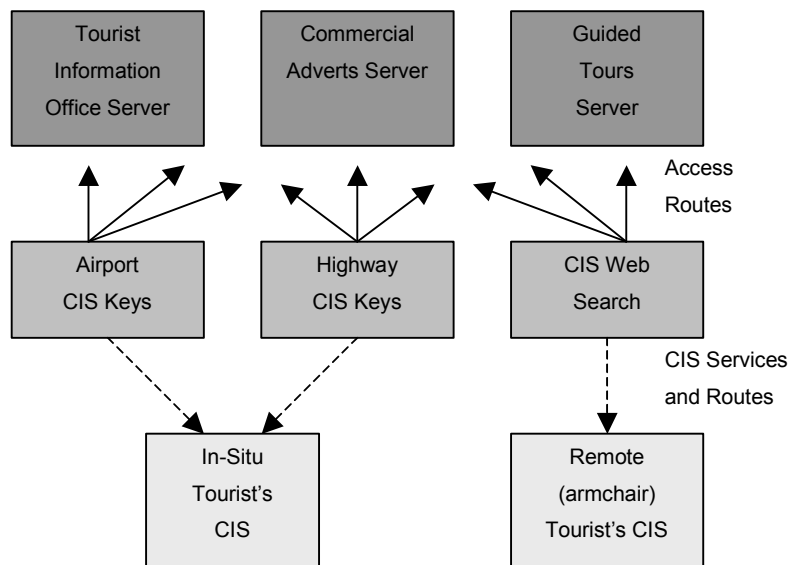
Figure 43. Example high-level CIS distribution and discovery model.

Figure 43 illustrates a simplified example of three physically separate CISs that may each be of interest to a tourist. Both an armchair and in-situ tourist can gain access to these CISs, though they will locate them in different ways and will probably use a different communications medium to contact them.

Some CIS worlds may not contain artifacts with a location context element, in which case a spatially situated CIS key may at first seem irrelevant. However, the role of the CIS key is to provide route information to CISs and is not solely for situating CISs that are describing entities in the current locale. For example, a CIS key may inform the

tourist's CIS of a cheaper and faster route to one of the remote CISs that her CIS is currently connected too.

8.2.2   Choosing Distribution and Managing CIS Loading

There are a number of ways in which the CIS world could be distributed ranging from a subscription service, in which local copies are kept and updated, to a fully distributed service where remote objects are accessed directly. The latter case is most suited to a CIS due to the requirement for real-time modelling of context. The CIS would provide transparent access not only to the artifacts within its own world but also to those in remote CIS worlds. When accessing the contents of a remote artifact the local CIS would work directly with that artifact rather than a local copy. Such an arrangement provides the ultimate timeliness/accuracy for modelling and minimises local storage space requirements. However, it does face potential problems with overloading CISs that are acting as servers (e.g. a tourist information office's CIS) and with coping with limited connectivity (e.g. a costly mobile phone connection from a tourist's handheld computer).

There will undoubtedly be situations where one or more CISs are loaded far more heavily than the others are. This is probably the case with the tourism application where there could be a tourist information office CIS that can expect to be accessed by many electronic tourist guides simultaneously. Not only is there a likelihood of heavy loading but it is also difficult to predict when that heavy loading will occur, e.g. 100 tourists may unexpectedly arrive at once. Therefore, the CIS must be able to dynamically respond to varying load conditions.

If a CIS world is being overloaded with remote accesses then it needs to be segmented to try to divide the remote interactions and share them among more than one computer to cope with the load. It is likely that for large systems this will already occur to a limited extent in order to logically group artifacts together in different worlds. For example, the tourist information office may have all the town attractions on computer X, all the adverts on computer Y, etc., and a unified world presented through a federation of the CISs. This form of underlying segmentation is mainly for organisational or management purposes and it doesn't effectively combat the problem

of one machine getting overloaded, e.g. lots of systems accessing the computer dedicated to tourist attractions.

A *CIS replicator* can solve the overloading problem by taking responsibility for providing accessibility to one or more worlds to the rest of the federation. A CIS can hand off all accessing responsibilities to the replicator, which keeps a duplicate copy of the world that the CIS is responsible for. The replicator is updated by the CIS whenever its world changes (as the original CIS still retains responsibility for the management of the world, it is just the external accesses that are delegated). Other CISs subsequently communicate directly with replicator but can still rely on completely up-to-date data.

Should the replicator become overloaded then it can hand off some of its responsibilities to child replicators (a process transparent to the source CIS). And should these child replicators become overloaded then they too can hand off to further replicators ad infinitum. In this way a tree structure of replicators is dynamically constructed (as illustrated in Figure 44) as and when the need arises, with each replicator managing its own loading and passing of updates to children. A linear array of replicators could also be used but the tree structure allows for a delegated 'chain of command' for managing the addition and removal of replicators.
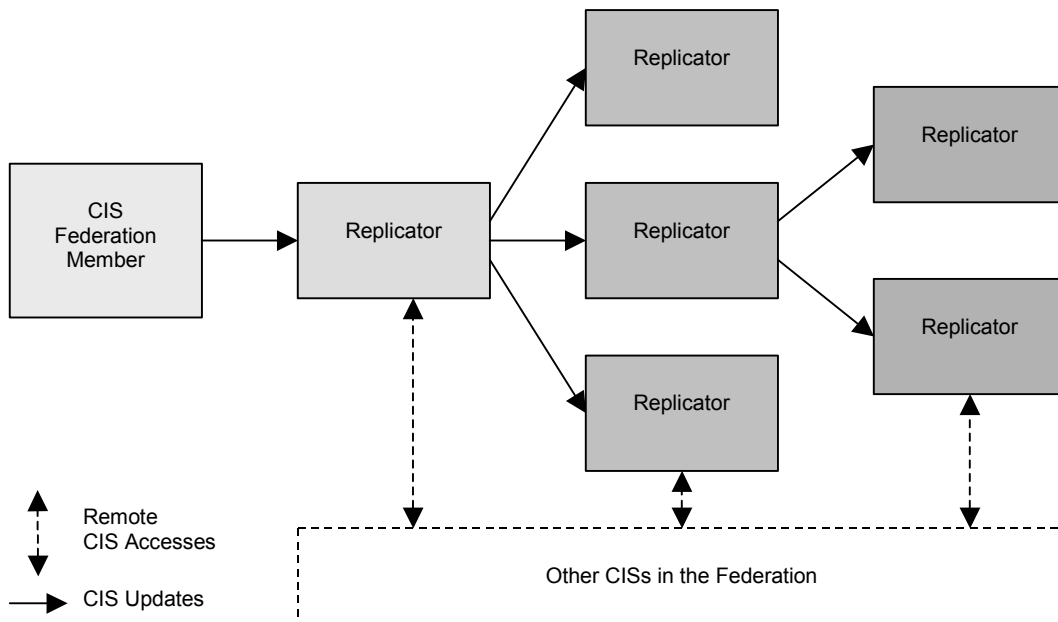
Figure 44. Three levels of CIS replication.

The replicator model provides a flexible and scalable way of dynamically coping with the overloading of a CIS world caused by an increase in accesses by external CISs. However, there are, of course, standard distributed systems issues associated with them, e.g. potential for uncontrolled expansion of replicators, problems when a master crashes, etc. There are also two other potential sources of overloading: sensor updates and federation maintenance messages, but these are much less likely to stress the system. However, if they do then sensor updates can be internally regulated by the CIS. Dealing with federation maintenance message overload is covered in a later section.

8.2.3 The Structure and Formation of a Federation

A federation is initially formed by a CIS that specifies a remote CIS it wishes to federate with and a name to identify the resulting federation. The federation is negotiated on a peer-to-peer level where each CIS is asked what level of connectivity it would like with the other. Each CIS responds by specifying one of five levels of connectivity it would like:

➢ **Reject**. A reject is returned if the local CIS wants absolutely nothing to do with the remote one, in which case the federation process is aborted.

➢ **Passive**. The local CIS accepts the federation but neither wants to observe the remote CIS's world or have its own local world observed.

➢ **Observer**. The local CIS is only interested in observing the remote CIS's world, and not in sharing its own local world.

➢ **Provider**. The local CIS is willing to provide its own local world to the remote CIS but does not want to observe the remote CIS's world.

➢ **Full**. The local CIS is interested both in being a provider and an observer of the federated CIS world.

Other members may then be inserted into the federation through a similar process, where every CIS in the federation negotiates its own connectivity with the newcomer. In the process of joining the federation, should one of the federation's members issue a reject then the newcomer is completely barred from the federation no matter how other

members respond. If the member successfully joins the federation it will communicate directly with each member rather than through its original point of contact (such a scheme would lead to uneven distribution and bottlenecks). Another method of forming federations is by merging two federations together. In this process a new common name is agreed for the federation and each CIS in one federation negotiates with each CIS in the other. The merging produces a new federation with no knowledge of the previous two federations (and hence, no way to automatically revert to the previous configuration).

Each CIS records information about the federations that it is a member of at three levels:

➢ Firstly, each federation has a name that is used to differentiate it should a CIS join more than one federation. However, to the CIS client a single world is provided transparent of the underlying federations (although there may be explicit mechanisms made available to query or filter by federation).

➢ Secondly, each member of the federation maintains a list of all other members of the federation and the type of connectivity between them. Everyone will have the same CISs in their list but not necessarily the same type of connectivity. Thus, each member may have different views on to the federated world.

➢ Finally, the CIS inserts references into its world of all the remote artifacts it can see within the federation. Federation members subsequently keep each other appraised of artifacts arriving and departing. Note that the reference is a mechanism through which all queries and manipulations are translated to the remote source, ensuring accurate up-to-date data and minimum storage requirements on the local CIS.

To create and maintain this federation information a number of messages are passed between members. These messages fall into three categories:

➢ **Membership Messages:** join requests, merge requests, leaving notifications.

➢ **Formation Messages:** list of artifacts sent to a new CIS, list of artifacts received from a new CIS, connectivity requests (to dynamically renegotiate the connectivity with another CIS).

➢ **World Maintenance Messages:** artifacts added, artifacts removed.

Such message passing is another potential source of overloading for a CIS, e.g. a CIS being unable to cope with all the messages about the artifacts coming and going from another CIS in the federation. This is the reason that several levels of connectivity with other CISs are provided. A local CIS may decide to be solely a provider to another CIS and not an observer so that it will not receive these maintenance messages. At the extreme it may wish to have no contact at all with the CIS and so specify a passive link. Even though there is no communication between the CISs in such a case it is still important to have a passive link. This is so that when a new member joins the federation using a particular CIS as an initial point of contact it can be informed of all the other members of the federation irrespective of whether the initial point-of-contact actively communicates with them or not. As each federation member is aware of every other (even if only through a passive link) a new member may join the federation using any member CIS as an initial point of contact.

8.2.4   Security and Privacy

A federation is an essentially open model where, in general, the individual CIS members openly share their world of artifacts amongst the other members. However, for reasons of privacy and the potentially sensitive or commercial nature of the data, a CIS may wish to control access over its world. There are three general levels of security:

- **World Level**. This is the coarsest granularity of security whereby a CIS may wish to restrict the access to its world as a whole. The mechanism for doing this has already been outlined: when joining a federation accessibility is negotiated between the new CIS and each member of the federation to determine which CIS worlds the newcomer can view and which CISs it will let view its own world. Note that different access rights can be specified for each federation member.

- **Artifact Level**. At a finer granularity the CIS may not wish to share individual artifacts within its world with particular CISs. For example, perhaps the user does not want to reveal information about herself that she has modelled in a 'user' artifact.

- **State Level**. At the finest level of granularity individual states within an artifact may be declared as private so that although the artifact in which they are embedded may be publicly accessible throughout a federation that particular state is not. This level of security is particularly useful in giving a degree of anonymity to a shared artifact. For example, in the tourist guide application the tourist may be willing to share the artifact representing herself but wish to keep the 'tourist name' state private. This will allow other CISs to see where tourists are and what they are doing but they will not be able to associate this information with specific tourists, thus preventing 'big brother' scenarios in which a central CIS is used to spy on an individual.
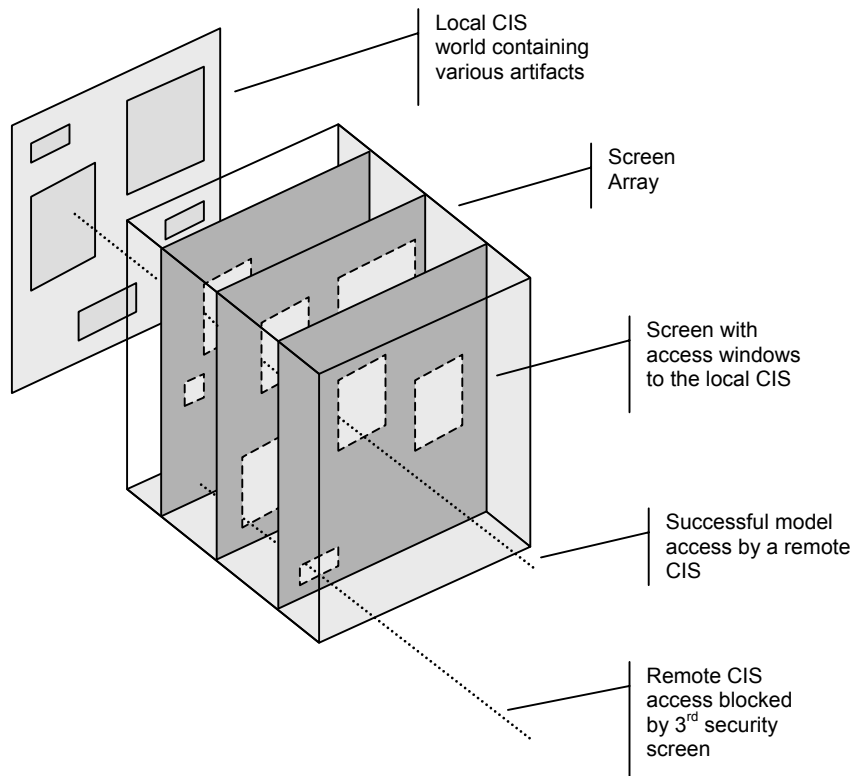
Figure 45. Conceptual diagram of security screening.

All three levels of security can be provided through a single *security screener* service. A security screener uses an array of screens to regulate access to the single CIS that it is responsible for. For each remote CIS that the local CIS is federated with, an array of screens can be constructed. Both incoming and outgoing communications are filtered through this array to block unauthorised communication. Each remote CIS is allocated

a screen array in the local security screener, and the slots of this array can be filled with screens from a reusable collection. A single screen array is illustrated in Figure 45, showing how communication from a remote client is blocked unless there is a window right through all the screens on to the part of the world that it is interested in. A screen is essentially a filter that defines which artifacts and states within the CIS world are and are not accessible through it. In Figure 45 the filtering was illustrated as windows in a succession of screens. However, the filtering can also be specified from an inverse perspective, which, in terms of the diagram, would appear as patches through transparent screens instead of windows through opaque screens. Patches or windows can be created to cover areas ranging from single states to single artifacts or even the whole world. Once defined, a screen is placed in a reusable pool that can then be used to populate one or more screen arrays. In addition to the screen arrays, for each remote client there is also a universal array which all communications must pass through. A communication must pass successfully through each screen in both the remote CIS array and the universal array before being transmitted. This allows universal and CIS specific security policies to be implemented simultaneously.

An interesting possibility for security screens is the notion of blurry windows. Rather than two simple can-see-it / cannot-see-it states, blurry windows could let a state be seen but would cause the value to be blurred, e.g. the location could only be distinguished to within a 100 metre area rather than a pinpoint location. This could be useful in applications where general low-resolution information can be revealed but more accurate information needs to be concealed. For example, a person may allow others to see that she is in town but not disclose the exact place.

The more security restrictions placed on the CIS, the greater the processing overhead in communications with remote CISs; this is unavoidable. However, for a completely open CIS with no security restrictions the security infrastructure does not pose any obstacle, i.e. there is almost no operational overhead in having the security screener with no screens. Additionally, in implementing a security policy the order of the screens could be managed to optimise efficiency, e.g. a screen that blocked out 99% of the world could be placed first to eliminate the bulk of communications before they get to the remaining screens.

The security screener could also be a good way of implementing access charges. Instead of simply a clear path through a series of windows, the path may also pass through 'coloured' windows indicating that a certain amount should be paid for such a communication. In a commercial CIS service clients could therefore be charged different amounts for accessing different parts of the world. However, as with any other service that provides data in digital form, how to prevent customers from freely passing data on to others must be considered. We do not attempt to address this general, but complex, issue here.

The CIS federation security is partially built on the trust of other members. It is therefore impossible to guarantee the security of information once it is even partially made available to others unless we completely trust the remote CISs concerned. For example, if CIS A gives access of an artifact to CIS B, although A prevents others from accessing the same information directly, there is nothing stopping B, if he is unscrupulous, to act as a gateway for unauthorised access to this information by providing a local copy to others. The only way to be completely secure is to keep the relevant parts of the world private to the local CIS.

8.2.5 Managing Poor Connectivity with Windows and Mirrors

The distributed world set forward thus far has assumed a high speed TCP/IP connection between the members of the federation and has not taken into account the poor network connectivity that some members may face, e.g. a tourist's handheld computer communicating via a mobile phone. Providing direct access to a remote world across the federation is ideal for maximising the timeliness of data and minimising local memory and storage requirements, but some of the members could have connections that make communicating over low bandwidth and dealing with periods of disconnection more pressing issues. Such connections could exhibit some of the following characteristics:

➤ **Unreliable**. The communications may be susceptible to errors and/or periods of disconnection.

➤ **Low bandwidth**. The throughput of information may be limited.

➤ **Costly**. Connection time can be expensive for devices such as mobile phones.

➢ **No TCP/IP**. Rather than a Java RMI-friendly TCP/IP protocol there may just be a plain serial link.

➢ **Point to point**. The communication medium may be point to point rather than global.

The latter two characteristics of no TCP/IP and point to point communications need to be addressed at the communications protocol level. CIS connections with these characteristics can no longer rely on Java RMI to handle the distributed communications; instead an alternative low-level form of communication must be devised. This alternative form of communication can be provided through a *CIS connector*. Rather than talking directly to a remote CIS, communications now pass through this CIS connector that provides an interface for membership, formation, and world maintenance messages to be transmitted to and received from a CIS connector of a remote CIS. Additionally, it provides methods through which artifacts and states of a remote CIS world can be accessed over these poor connections. Note that as a CIS connector is required on each side of the connection: even CISs with high-bandwidth network access will have one in order to talk to a CIS with poor connections.

The connector has no notion of the state or membership of a federation; the CIS retains the knowledge of where a remote CIS is, what artifacts it has, and how to contact it. The CIS connector simply acts as a courier responsible for transparently delivering CIS messages and data between two CISs over a poor connection.

The combination of high-cost, low bandwidth, and unreliable communication characteristics requires a radically different approach than in the constantly connected networked CIS. Communications need to be not only reduced but also made more flexible to deal with unreliable communications and slow (or non-existent) turn around times. The CIS connector addresses these concerns with two strategies:

➢ **Access Windows**. To flexibly minimise/optimise connection use.

➢ **Artifact & State Mirrors**. To attempt to cope with periods of disconnection.

Access windows are periods of connection time through which a CIS connector is able to access a remote CIS connector. These windows can be flexibly defined to balance

the need to optimise communications usage with the remote access requirements of the CIS. A CIS connector will use one or more of these windows to implement its remote communications strategy. A window consists of an opening and a closure that define the start conditions and termination conditions of a period of communications. There are three types of opening and closure with which to define a window, as illustrated in Table 29.

	Opening	Closure
Scheduled	The opening is scheduled for a regular repeating interval, e.g. every 10 minutes.	The window is closed after a predetermined time period irrespective of whether the link is still being used or not in the meantime.
Dynamic	The opening is dynamically created when the need arises, e.g. when a CIS attempts to access a remote artifact.	The window is closed as soon as it is no longer being utilised.
Delayed Dynamic	As above, with the addition of a predefined delay before the opening in order to wait for other remote communications that can be batched together.	The window is closed when a predefined time period after the previous communication expires. This allows other communications to arrive in the meantime (after which the period is reset).

Table 29. Window opening and closure types.

A window can be created from any pairing of an opening and closure type, e.g. a dynamic opening combined with a scheduled closure. Additionally, openings or closures may be combined so that an opening or closure will occur when any of the individual conditions are met. For example, a dynamic delayed closure could be combined with a scheduled closure, where if the delayed closure does not occur before a certain time period then the scheduled closure will. Such a scheme is useful for putting a maximum time limit on a window that will normally have a dynamic delayed closure. Figure 46 illustrates some simple window schemes.

The CIS connector will implement its communication strategy by defining a window for each communication channel it has access to. In some rarer cases there may be the need for more than one window per channel, for example, to temporarily increase the communications capacity by introducing another scheduled window, which will later be removed.

Before a window is opened the remote connector must give its consent for the process to proceed. Normally consent will be given because the remote CIS is presumably

interested in the data that will be sent, and also the remote CIS can use this window to perform any additional communications that it needs with the local CIS (in effect, getting a free connection). However, in some situations the remote client may decline or may simply be unavailable, in which case the window is not opened and the local CIS is informed that it is currently not accessible.
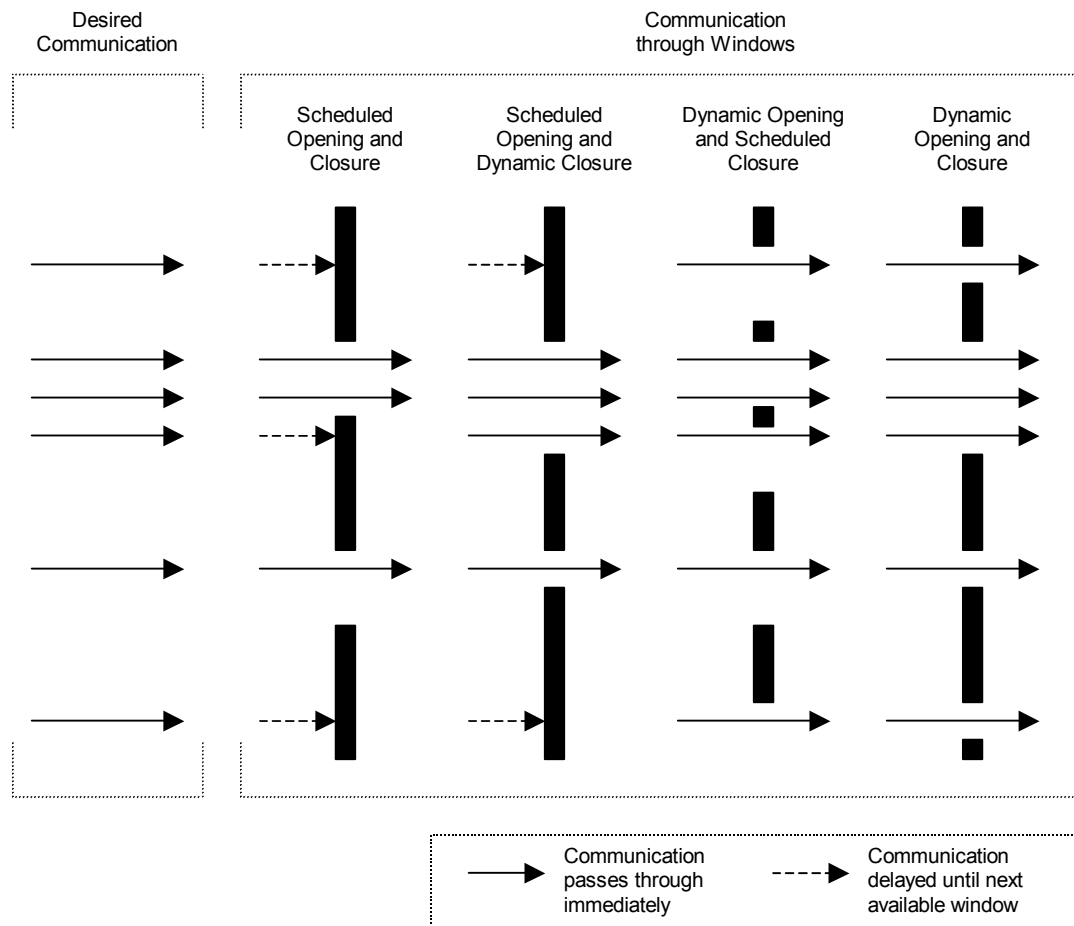
Figure 46. Example window schemes.

Unavailability and periods of disconnection are unavoidable with some communication mediums. The distributed CIS needs to cope with such conditions as best as it can (and fail gracefully when they become untenable). The CIS connector does this through an array of 'mirrors' that model individual states of a remote CIS's artifact locally, where one slot of an array can contain one mirrored state. The mirrors can then be used by the CIS connector as a substitute for the remote artifact states during periods of disconnection.

The array is dynamically partitioned into two areas, with one side that can be populated by the CIS client (who will be either a user or another piece of software) and the other side that can be populated by the CIS connector. A user may add a mirror to the array to ensure that a remote state that is essential to their successful operation is always available (albeit, perhaps, in an out of date form). The CIS connector is also trying to ensure the same thing by automatically adding mirrors to the array in order to locally model remote states that are heavily used by CIS clients. The partitioning of the mirror array is dynamically controlled so as to maximise space utilisation.
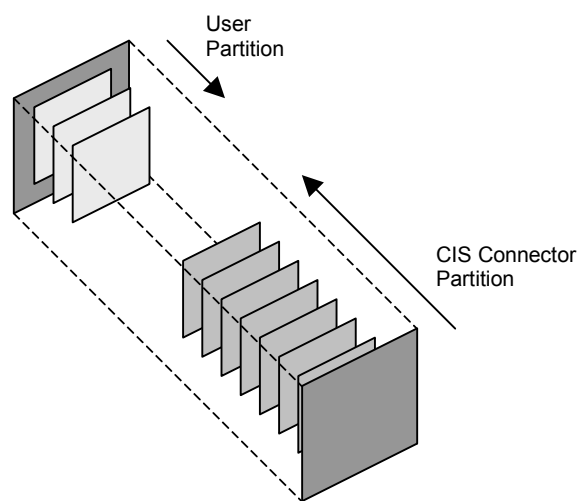
Figure 47. Conceptual view of a mirror array

A mirror is created by duplicating a state from a remote artifact. However, in addition to its current state it may also obtain *drift* or *prediction data*. The drift data expresses how the accuracy of the state decreases over time, whereas the prediction data supplies information on how the state will change over time and the confidence in those predictions. Rather than a static snapshot of the remote source, predictive data enables the local CIS connector to attempt to locally mirror the changes occurring to the remote state. Of course, the CIS client is informed that mirrors are being used rather than real data and is given the potential inaccuracy of the data if drift or predictive information is available. When the remote CIS is directly available again the mirror is updated each time the remote state is accessed, therefore placing no added load on the communications medium to maintain the mirror (except when initially created). A client may also instruct a mirror to autonomously refresh itself at regular intervals. This is intended for state mirrors that are essential to the operation of the client but which are

infrequently accessed and therefore not otherwise kept up to date enough. The autonomous updates will occur in periods when there is an access window available.

In some cases the client may be able to use an alternative source for getting information, e.g. when the national weather forecast is not accessible, it uses its own barometer in order to make a guess. For this reason it is useful for the client to be able to know when a mirror is being used rather than the real source, and what the likelihood of error is.

8.2.6   Federation Design Summary

A CIS federation provides a means to distribute a CIS world over many physical devices whose own CISs are each responsible for the maintenance of their part of the shared world. The membership of the federation is agreed upon by all the members, although some CISs may express more interest in a specific area of the federation than others. Should the federation start to overload any one CIS in particular, the CIS can delegate its communication responsibilities to a replicator hierarchy that dynamically manages this external interaction load. If a CIS member is poorly connected to the rest of the federation it can channel its communications through a CIS connector that implements a communications strategy based on the characteristics of the connection. Through such mechanisms the federation attempts to distribute the CIS world in a way that takes into consideration overloading, connection characteristics (time, cost, bandwidth, etc.), storage requirements, and timeliness/accuracy of the contextual data. Although thought of specifically for a CIS environment, there are similar general-purpose mechanisms to be found in the general distributed systems literature [Tanenbaum 1994 - #140] [Slowman 1987 - #131] as is the case with the other concepts presented in this chapter.

8.3   The Logical World Distribution Design Concept

The federation concept is well suited in coupling CIS worlds residing on different devices into one seamless service whose underlying distribution is transparent to a client. The resulting CIS world is presented to clients as one unified group of artifacts that are the same in appearance as if they were residing on the same physical device. CIS members of the federation collaborate in a peer-to-peer manner in order to

maintain this distributed world. Such a distribution concept works well for a collection of equal status CISs combining to form a super-CIS world, but it does not work as well for collections of CISs whose individual roles are not in equilibrium. For example, a tourist guide application may have a more client-server nature. The tourist office's CIS may provide contextual information to the tourist (although the tourist clients may also contribute information about themselves) and the tourist's CIS may only be interested in viewing a specific subset of the federation members (e.g. just the tourist office's CIS). The tourist's CIS is probably not at all interested in the creation and maintenance processes, and consequent overhead, involved in being a federation member. Additionally, although the tourist CIS may only be interested in only a few specific artifacts within the federation it will receive information about all of them. The security screening mechanism can 'hide' artifacts from the federation but this can only be used to restrict what artifacts a member provides rather than allowing another member to express ones that it is interested in observing. For example, in joining a federation with the aim to communicate with the tourist board's CIS the tourist's CIS will also be directed to negotiate links with all the other members of the federation (such as all the other tourists) which is certainly not very desirable.

This tourist application example highlights the need for a distributed CIS world that can be consulted without necessarily becoming a full-blown member to do it. Tourist CISs are primarily concerned with viewing a collection of CISs containing tourist information rather than taking part in the management of such a collection. However, the limitations are not just on the client side. Take, for example, a tourist CIS federation that contains details of artifacts in London. Such a federation would be fine if the information were held on a set of CISs that contained only the information that was of interest to London. However, perhaps ABC cinemas have a CIS containing artifacts for each of their cinemas across the country. It would be extremely inefficient to have to join the London tourist federation simply to contribute a single cinema artifact to it. To summarise, federations are good at combining CISs that consist of entirely related worlds or artifacts but not for partial or customised groupings or for groupings of members with logically unequal roles.

A logical world provides an alternative distribution concept for applications in which a federation is unsuitable. Logical worlds support passive viewers and work at the level of

sharing artifacts rather than sharing worlds. Instead of a member CIS contributing all the artifacts in the world that it sees fit too, the logical world filters out the artifacts that are of interest or have a common theme or grouping. The idea of logical worlds is based on the premise that there will be sets of logically related artifacts that can be defined in advance of a need, and that these will be useful to more than one client.

8.3.1   A Layered Model

The logical world contains a related set of artifacts that are drawn from various other stand-alone or federated worlds. Note that the logical worlds are not a replacement for federations: they offer a different and complementary service. Logical worlds could be compared to views onto a relational database; they present a window onto a subset of interesting artifacts taken from a number of stand-alone or federated CIS worlds, as shown in the Figure 48.
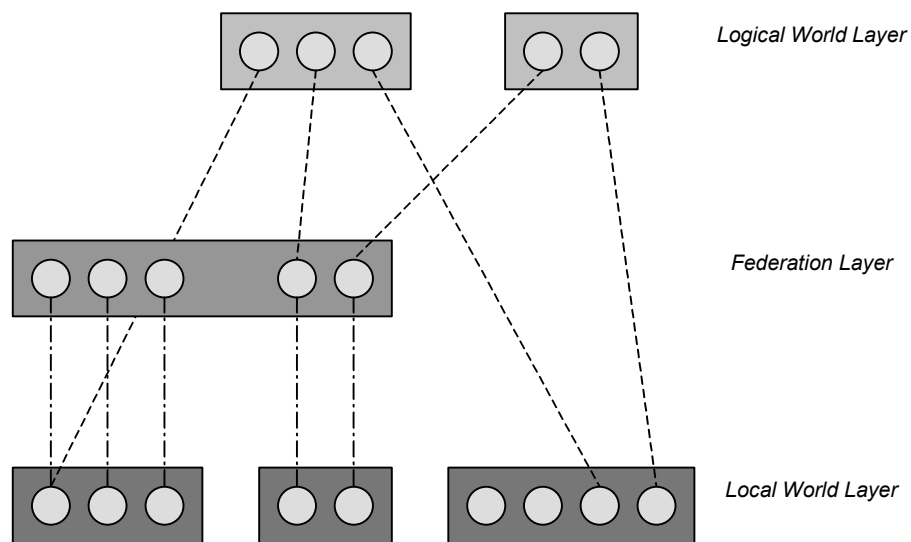
Figure 48. The logical world draws on the artifacts of federated and local worlds for its membership.

The figure illustrates how the underlying local worlds, the federated worlds, and the logical worlds exist as a series of conceptual layers in the CIS, with increasingly abstract groupings with each higher level. The local world simply supports the artifacts that the physical host device is aware of. The layer above allows federations of CISs to be formed where CISs collaborate to provide a shared world. Finally, the logical layer allows logical worlds to be formed that pertain to particular logical groupings of

artifacts. At the local and federation levels a CIS presents one unified world but at the logical layer there may be many different worlds presented.

8.3.2   Logical World Structure

The management of a logical world may be distributed across a number of CISs. This management distribution may reflect the underlying distribution of the member artifacts in local CISs. However, it could equally be undertaken by CISs that are responsible for managing the world membership of artifacts that are not from their own local world. In this way we have a very flexible way of distributing the management load for the logical world. Additionally, instead of dividing the same activities between CISs, different CISs may be responsible for different management duties. These duties typically fall into two categories: world maintenance (managing the underlying artifact membership) and world viewing (managing the viewing of the world by clients of the CIS). Therefore, some managing CISs may undertake all responsibilities, some may only assist in managing the membership, and others may just act as windows for clients to look into the world. This structure results in three levels of distribution:

➢ Management distribution.

➢ Client viewing distribution.

➢ Underlying artifact distribution.

A lightweight CIS could be constructed by utilising the logical world distribution. It could simply manage the viewing of a remotely maintained logical world, i.e. provide a world to its clients but take no responsibility for maintaining it. The processing load would be substantially reduced and the sensor array and catalog components could be removed from the CIS altogether (the archive may be kept if the client wishes to do any archival work).

Unlike federations, where different CIS members could negotiate different views onto the world, the logical world by its very nature presents the same view from any of the managing CISs (if they have undertaken viewing responsibilities, of course).

With the distributed management of membership there is the possibility that the same artifact can be added to the world by different managers. To cope with this the list of member artifacts that each manager maintains is also annotated with the parties interested in each artifact, i.e. the manager or client that added the artifact. The artifact is then not removed from the world until all the registered interests are removed (or if the artifact is removed from the underlying source world).

8.3.3 Creating and Maintaining a Logical World

To form a group a description of its membership has to be defined. The easiest way to achieve this is for the clients to explicitly define the individual artifacts that are of interest. However, for most purposes it would be preferable to specify the membership in a more abstract way. For example, people in the same room as me, all the white-knuckle rides in a theme park, etc. This definition is dynamic as it refers to states of an artifact that could change; therefore the artifact membership may also be continually changing. To support such dynamic membership the client must also specify the potential membership pool in the definition, i.e. those worlds whose artifacts the CIS should check when evaluating the membership. The definition of a logical world thus comprises the following elements:

➢ Logical world name.

➢ Static artifact member list.

➢ Dynamic membership conditions.

➢ Source worlds.

The process of creating and maintaining a logical world given this definition is now examined. Firstly, by considering the issues involved in a stand-alone localised logical world (albeit with artifacts from remote sources) and, secondly, considering a distributed logical world.

8.3.3.1 Making a Local Logical World

First of all the CIS creates a new logical world and populates it with the list of static members. It then evaluates the dynamic membership by matching the artifacts of each

of the source worlds with the set of dynamic membership conditions, e.g. all people whose location is within 50 metres of my office. The formation of the logical world is now complete.

Keeping the world up-to-date after its creation involves re-evaluating its dynamic membership. This could be carried out in a top-down fashion, where the logical world continually queries the underlying worlds, or in a bottom-up fashion, where the underlying worlds inform the logical world of any new matching members or existing members that should now be removed. However, to prevent over-loading the underlying worlds (especially likely if dealing with continually changing states such as location) a top-down approach is taken that places the burden of processing on the logical world. Therefore, the logical world needs to poll the underlying local worlds at intervals to re-evaluate the dynamic membership.

The polling interval could be defined as a set period after which the whole dynamic membership of the logical world is re-evaluated. However, this is both time-consuming and often a wasteful approach because different artifacts are liable to change at different rates. What is required is a more targeted form of polling that permits different polling intervals for different artifacts. For example, people every hour, cars every five minutes etc. Different intervals can be targeted at specific artifacts, general types of artifacts, or even the whole world. There are also two special types of interval: *continuous*, for a constantly updated membership, and *never*, for members that never need to be updated (e.g. the location of the Pizza Hut is unlikely to change). Clients can also inspect when membership was last updated and if necessary call for an explicit update of a portion or all of the logical world. These mechanisms combined allow the client to define an updating strategy that is appropriate for both the accuracy required by the application and the amount of processing resources to expend.

The client is generally responsible for managing the static membership, i.e. artifacts explicitly added and removed from a group. However, the client may specify a lifetime when adding an artifact into the logical world and when this lifetime expires the CIS will automatically remove it. Without a specified lifetime the CIS leaves it in the logical world for as long as clients are interested in it.

8.3.3.2 *Making a Distributed Logical World*

In addition to having the member artifacts from distributed sources the logical world itself may be managed in a distributed way. The same general processes and mechanisms explained in the local logical world are still used in a distributed logical world but with a few additional steps and mechanisms to allow for their distributed operation.

The main difference is that the logical group is created in sections rather than as one whole definition. When the client requests that a logical group be formed it can specify that it would like it to be distributed. This results in a repeating interaction with between CIS and the client to organise the distribution, as show below:

1. *The client informs its local CIS of the logical world it would like to create.*

2. *The client informs the local CIS of a CIS that it wishes to delegate some of the management and viewing duties to. The management duties may consist of only part of the overall definition of the logical world. Viewing duties consist of providing clients with access to the logical world.*

3. *The local CIS communicates with the specified CIS (if not itself) and issues the duty request.*

4. *If the request is accepted the client may move on to the next part of the distribution by returning to step 2 while there are still more duties to delegate. If not accepted the client may return to step 2 and try to assign the same duties to another CIS, or simply abort the process.*

This process establishes the initial distribution but this may be modified over time by adding and removing new CISs and altering the membership definition. For example, a client may decide to add a new CIS to take over some of the duties of another or to take on a new additional part of the membership definition. Merging of two logical worlds may also be performed to produce one common world, e.g. combining the 'people in office' and 'people in corridor' logical worlds into one 'people near my workplace' logical world.

8.3.4 Logical World Design Summary

The logical world provides a means to group a set of logically related artifacts that may be distributed across a number of underlying CIS local worlds and to present them to

clients as a single logical world. The grouping can be defined in abstract terms that allow members to join and leave the world dynamically as their states change. In addition to the distribution of the artifact members, the management processes of the logical group itself may also be distributed across a number of CISs.

These features allow for a much more sophisticated and flexible model of distribution than federations but one that is also much more complex.

8.4 The Self-Forming Distribution Design Concept

Both federations and logical worlds require some degree of participation by the clients in order to specify the distribution configuration. However, for many clients it would be desirable to simply use the local CIS without any participation in its underlying distribution and have it transparently and autonomously provide access to remote artifacts that are of specific interest to it. In essence, incorporating artifacts of interest to the local CIS as and when they are needed with no formal grouping infrastructure to manage.

Logical worlds are limited to groups of related artifacts that can be specified in advance and are of common interest to clients, e.g. a logical world of artifacts for Leicester Square. Instead of this the self-forming distribution model can dynamically add and remove remote artifacts from the local CIS world, tailoring it to be of direct interest to its clients and requiring no help from them in distribution and group set-up. There is a need for federations to support the low-level distribution of a large CIS over a number of computers, and there is also a need for logical worlds to support groupings of artifacts of general interest to many different clients. However, the self-forming distribution model addresses another area: that of providing a world whose transparently distributed artifact membership is dynamically and autonomously updated to reflect the current interests of its clients.

There are two main challenges in the design of a self-forming autonomous distribution: (1) finding remote worlds and (2) managing the transparent and dynamic incorporation of *relevant* artifacts from those remote worlds into the local CIS's world.

8.4.1 Finding Worlds

To incorporate remote artifacts into the local CIS world the first step has to be the discovery of remote worlds that form the source of these artifacts. CIS keys (as described in the earlier section on federations) provide an ideal method of remote CIS discovery. Thus far they have only been considered as in-situ contact information automatically presented by a device in order to discover connections to a remote CIS. An example was previously given of a tourist entering an airport whereupon his PDA detects a number of keys that are being wirelessly broadcast. This section shall go into more detail of the services they provide and how they provide them.

A key simply contains a description of how to contact a remote CIS, e.g. details of a dial-up connection. The important aspect of keys, however, is their embedded nature. A local CIS discovers the key when it is in a particular physical context such as the airport. Rather than a global centralised hierarchy of CISs that need to be looked up like a DNS, embedded keys allow access points to remote CISs to be situated and distributed in the contexts where they may be required. The key is physically situated in a particular contextual space and can be discovered by a CIS when it also present in that space. Of course, it is then up to the local CIS to decided if it wishes to use the remote CIS or not.

Keys may be embedded equally well in both the real and virtual world. For example, a key may be placed at the physical entrance to Alton Towers but also embedded as meta-data to be discovered when examining the theme park's web page (in effect linking content to context). To support the discovery of keys in different contextual spaces different technologies can be employed. To embed the key in a physical contextual space (e.g. at the entrance to the theme park) some form of 'black box' device may be physically embedded in the pavement to wirelessly transmit the keys (e.g. via Bluetooth). Note that the contextual space of the black box and the key are not necessarily the same, e.g. the key may exist in the same location space but only be present during 9am-7pm. For more virtual and abstract contextual spaces such as a Web page there may be no need for any 'key server' technology as the CIS can just search the viewed Web pages for keys. This will require some explicit effort on the part of the local CIS. Virtual objects such as applications (including web browsers) and system processes could also present keys to the CIS through some 'plug-in' type

mechanism. Finally, the local CIS itself may contain some keys that it uses when their context matches that of some other specified artifact. In this case the keys are discovered by an internal process rather than being presented by an external force.

An interesting aspect of using physical 'black-box' devices to transmit keys is that their location may be relative or proximity-based rather than absolute. For example, rather than attached to the entrance of Alton Towers, a black-box could be embedded in a moving object such as a bus, robot, cruise ship, printer, dog, tank, person, etc. Keys to the CISs of these devices could therefore be discovered when coming within proximity of them (or even in contact of them). It maybe that the CIS for, say, a person, resides on some network far away but they have a little black-box attached to them that transmits the key (perhaps in the form of the CIS's network address) to other persons' or devices' CISs within range of the transmission. To the device or person the remote artifacts describing the person become available transparently as they approach them.
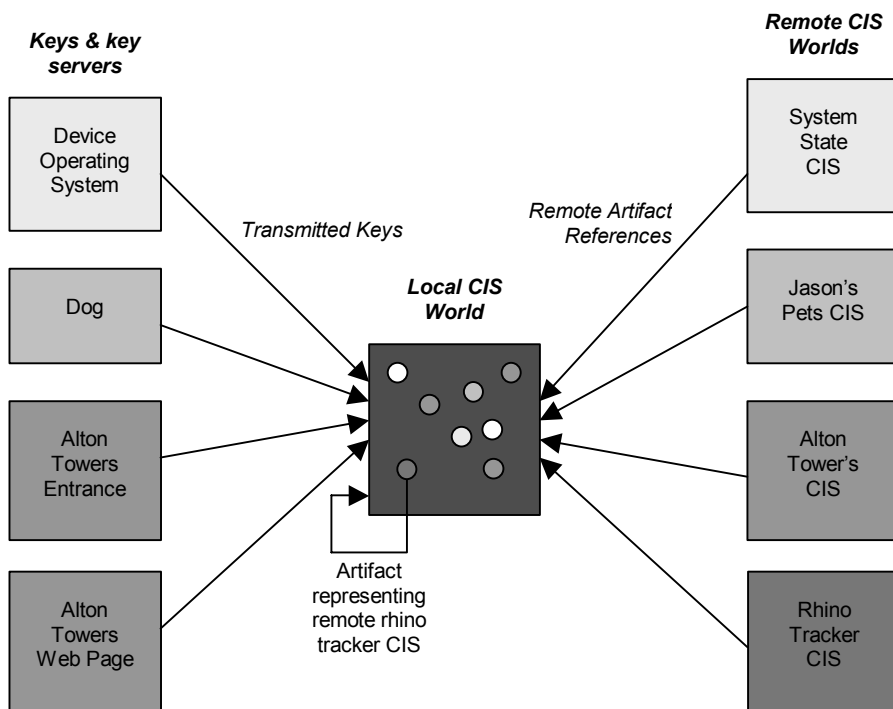
Figure 49. Keys from a variety of sources are discovered and used as a pool of artifacts from which ones of interest are entered into the local world in the form of a reference.

In summary, embedded CIS keys act as filtering and grouping mechanisms that allow remote CIS worlds to be discovered when and where they are needed. They can be thought of as a multi-contextual indexing mechanism into a planet filled with CIS

worlds, where the current index is derived from the current contextual space of the local CIS. Figure 49 illustrates the discovery of keys and the ensuing flow of interesting artifacts from the remote CISs.

The key servers are totally distributed with probably no connection to any other system except for clients who enter the same context. In this way the system can be infinitely scaled as there is no performance impact as new key servers are added (because they are disconnected from anything else other than clients in the same context space).

8.4.2   Incorporating Relevant Artifacts

Finding the remote CISs is only the first stage of formation in a self-forming group. The next stage involves selecting artifacts that are of interest or relevance in the remote world and transparently adding references to them in the local CIS world. Simply adding all of the artifacts from the remote world is not a good idea as the local world would quickly fill up with references of no use whatsoever. However, to select artifacts of relevance the local CIS needs to know what its clients' interests are. There are two methods of doing this:

i.   *Home Artifacts.* When creating an artifact template the designer can specify a profile of what is of interest to that artifact, e.g. for a person artifact template an interest may be specified in all things within 20 metres. When an artifact is created from a template and added to a CIS world its interests are normally ignored. However, there may be certain artifacts within the world that represent the entities that CIS clients are working for, e.g. the 'user' artifact. These artifacts can be nominated as home artifacts and as such their interests will be used in determining which of remote artifacts will be incorporated into the local CIS.

ii.   *Explicit Interests.* Some clients may have temporary interests or simply interests that are not related to any artifacts represented in the local CIS world. In such cases they can register an explicit interest that is the same in form and function as those used for home artifacts.

Both types of interest are essentially a form of query that use artifacts, types of artifacts, or states of artifacts as their terms. These interests may also be conditional, i.e. the

interest only exists under certain circumstances. For example, a tourist artifact may be interested in restaurants if lunchtime is approaching and she has made no reservations as yet.

After discovering a key to a remote CIS the local CIS compares the explicit and home artifact interests with the artifacts in the remote world. Any remote artifacts that match an interest are stored as references in the local CIS world. These references are linked to the interests that brought them to the local world, and if all interests in a reference disappear then it is removed from the world. This process is relatively straightforward, but the maintenance of the remote references due to changing interests and artifact states is more complex.

To maintain references to distributed artifacts, as the states of artifacts in the remote worlds that the local CIS knows of change, local references should be added or removed as appropriate for the current interests. The local CIS must undertake the responsibility for detecting the state changes in the remote artifacts because the remote CIS probably has no interest in the matter. This implies that the local CIS must intermittently check the remote artifacts; a polling system rather than the external event-driven nature of remote world discovery. When and how much an artifact changes is unpredictable and hence leads to the problem of deciding when to check the remote world. Polling at fixed intervals is a poor solution as each artifact may have vastly different rates of change; also the clients may require different levels of accuracy. To solve these problems an update interval is supplied with each of the interests so that the clients can define what granularity of accuracy they require on membership of different artifacts, e.g. busses could have an update interval of 30 seconds. The interval specifies the time between evaluations of an interest so that there is a guaranteed period of rest, i.e. the interval is timed from the end of the last evaluation rather than when it started (hence, avoiding overlapping evaluations). A reference is time-stamped with the time of its last evaluation so that clients can determine the currency of its membership (especially useful if there is more than one interest for a reference).

Another maintenance problem is the build up of discovered keys as the CIS moves around the world. Each key that the CIS finds is used to add another remote CIS to the list of those checked when evaluating interests. If keys are not disposed of when no longer of use the local CIS will quickly become deluged with hundreds and thousands

of remote artifacts it needs to check when evaluating interests. Therefore, there certainly needs to be a way of disposing of keys when they are no longer needed. But how does the CIS judge when a key is no longer needed? The number of remote references to that key is not a valid indicator, especially if artifacts frequently come and go from the remote CIS. A better method is to define the contextual space of the whole remote CIS itself. This contextual space can be transmitted as part of the key and subsequently used by a local CIS to compare with its own interests. If there is no overlap between the local interests and the contextual space of the remote CIS then the remote CIS and key can be thrown away. This overlap checking can be 'piggy-backed' onto the evaluations of each interest, with only a minor processing cost equivalent to evaluating the membership of one more remote artifact. We can therefore ensure that there are no redundant keys being kept.

If local CISs move directly out of the contextual space of a remote CIS this solution works well. However, if they tend to hover around the edge of the contextual space it will cause the client to be continually flipping between discovering and disposing of the key, which is liable to cause considerable processing overhead. This 'hovering effect' is caused by placing two transitions on one boundary, i.e. discovery and disposal occur on the same boundary but by crossing it in different directions. This problem can therefore be solved by creating two separate boundaries: one for discovery and one for disposal. Now a remote CIS will not be discovered until crossing the discovery boundary and will not be disposed of until exiting the disposal boundary. The discovery boundary describes the contextual space that the key inhabits in terms of a local CIS *discovering* it. The disposal boundary describes the contextual space that remote CIS inhabits in terms of a local CIS not *disposing* of it. There may also be a need for an extra space in which the key is not used but not disposed of either; a kind of buffer zone so that should the client re-enter the discovery space it already has the key to hand. This 'onion' of spaces described by a key is illustrated in Figure 50.

Ideally these spaces could be specified relative to the device on which the local CIS resides, therefore being equal to all devices. For example, five minutes from the space is more universal than 2 miles because a person may take 20 minutes to walk 2 miles whereas a car may take 1 minute. Specifying the *time-to-return* is universal for all objects. It could also be a useful measure in ignoring keys whose contextual space the local CIS

will only be within for a very short time. These indirect forms of specifying contextual distance make an interesting area for future research.
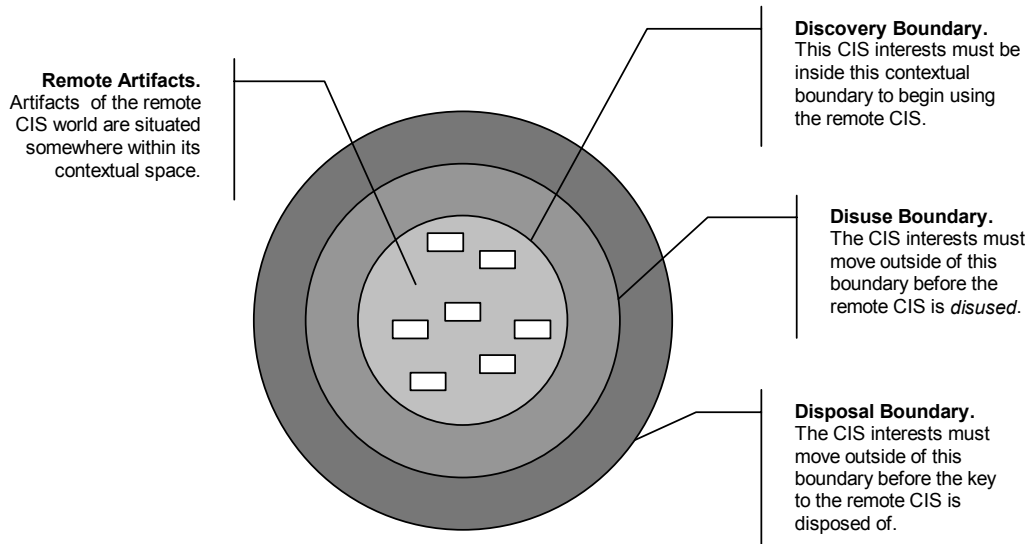
Figure 50. Contextual boundaries used in discovering, disusing, and disposing of CIS keys.

Note that the same "hovering effect" problem occurs in triggering, and the same contextual boundary solution can be applied to solve it.

8.4.3   An Agent Based Approach

The autonomous nature of the self-forming distribution model makes it well suited to employ software agent technology [Kalakota, Stallaert 1996 - #70]. The following sub-sections describe a set of three specialised agents that could work on behalf of a CIS in the provision of self-forming groups.

8.4.3.1   The Key Agent

The *key agent* continually works in the background, as do the others by their nature as agents, looking for keys in various places. The agent relies on a set of customisable modules that operate in specific domains, e.g. one for listening for infrared key broadcasts, another for examining a browser's current web page for embedded keys, etc. These domain modules can be turned on and off, new ones added, and current ones removed, and other concurrently running programs can dynamically register as modules.

In the tourist application where a tourist may wish to make her CIS available to the world, the key broadcast by her palmtop computer may refer interested parties to a replicator residing on her home PC rather than the local CIS on her palmtop. This delegates the external accessing of her CIS to a computer with more processing and communications resources than her limited palmtop device. The local CIS will update the replicator at given intervals and clients of the replicator can examine the time of the last update.

Rather than broadcasting access information to every person and device who cares to use it, some clients may want to restrict who can view the information. This is achieved by encrypting the contents of the CIS key so that only devices that have been given the appropriate public-key (in the encryption sense of the term) can decipher the contents and attempt to access the remote CIS. Others without the public-key will still discover the CIS key but be unable to use it. Even those who do successfully decrypt the key will face another level of security at the remote CIS in the form of the security screener.

8.4.3.2 The Interest Agent

The *interest agent* is responsible for continually maintaining the membership of references to remote artifacts within the local CIS world. The key agent passes the interest agent any keys that it discovers, which are then used by the interest agent to form the source pool of remote artifacts that it must check for membership. This membership evaluation is executed by spawning a new process for each interest present within the CIS. These timer-driven processes perform the evaluation at the rate specified by the interest's interval, checking all the artifacts from each remote CIS whose contextual discovery space they lie within. They also time-stamp introduced references each time they are evaluated so that clients can query the currency of their membership. If interests lie outside of a remote CIS's disuse boundary then the interest checking process does not use that remote CIS as part of the source pool of artifacts to evaluate (and references already present from that remote source are removed). The interest agent regularly downloads a new copy of the remote CISs contextual space definition as it is possible that it will change over time.

The interest agent monitors the interests and makes sure that if one disappears, if its conditions are no longer met, or if it changes form, that the related interest process and

references are removed or updated accordingly (taking into account multiple interests in a reference). If an interest is simply modified then the related process carries on as normal but the client can request that the next scheduled evaluation be brought forward (the interval remains the same thereafter).

The interest agent is also responsible for interest, or rather lack of interest, in keys and their appropriate disposal. As long as there is one interest inside the disposal boundary then the key is kept, but as soon as it detects that there are none remaining the interest agent disposes of the key. Clients interested in a particular remote artifact may place a *hold* on the reference so that even if it or its remote CIS are no longer of interest the agent will not remove it until the client withdraws the hold. A hold may also be placed on a key so that it is retained despite having no related interests: in effect its disposal boundary becomes infinite.

8.4.3.3   *The Cache Agent*

The cache agent uses the same basic scheme as devised for federations, whereby an algorithm based on the amount, frequency, and time of accesses to a remote state are used to provide an automated caching scheme. However, the cache agent could also exploit an importance attribute specified on each interest to help determine which remote states to cache, honing its search on poorly connected but important artifacts.

Should a remote state be unavailable when a client tries to access it, the cache agent intercedes with the cached value, which it annotates with the date it was last updated.

8.4.4   **Self-Forming Design Summary**

The self-forming distribution model is concerned with dynamically populating the local CIS world with references to remote artifacts that are likely to be of interest to the local CIS. The result is a transparently distributed world of artifacts that is tailored to the needs of the local CIS's clients. The autonomous maintenance of this distribution is carried out by three agents: (i) a discovery agent that finds remote CISs through a variety of methods, (ii) an interest agent that assesses which remote artifacts should be included as references in the local world, and (iii) a cache agent that attempts to store a local copy of important remote states for use in periods of disconnection.

8.5   Conclusion and Future Work

In the preceding two chapters we have presented the idea of a context information service and described a prototype implementation. In this final chapter of the thesis we have explored how the context information service could potentially be distributed, as we believe that for many context-aware applications the ability to access remote contextual information will not just be useful but essential. In this chapter we have proposed a number of design concepts for structuring the distribution of the context information service.

The first concept for a distributed CIS was based on the idea of connecting separate and remote CISs into one co-operating super-CIS, or *federation*. The federation provided a flexible way of managing a distributed but common world of artifacts, with internal and external access control mechanisms and a scheme for coping with the overloading of individual CIS members. Federations present a good method of distributing a unified service over physically separate devices, but they do not provide any facilities for working with sub-sets or groupings of related artifacts. The second concept, called logical worlds, addresses that problem by allowing a logical world to be defined that consists of artifacts of a common theme that are drawn from a number of remote CISs. The management of a logical world can also be distributed in a very flexible manner allowing different members to undertake different amounts and types of management duties. A logical world is a good way of collecting related remote artifacts into a common pool that a number of CISs are interested in and willing to manage. The final concept, a self-forming distribution, works on behalf of the interests of a single CIS rather than a collective of mixed interests. Through a set of three autonomous agents it seeks to provide the local CIS with access to all the remote artifacts it can discover (and that are of interest to the local CIS) in a completely transparent and autonomous way.

The three distribution concepts have different characteristics that will make them suitable for different types of applications. However, within the scope of this work we have only sought to establish the CIS distribution design concepts. The refinement of these concepts into more formal models, the development of prototypes, and their trials within different application areas, we leave to future work.

Chapter 9:

Summary and Conclusion

We propose that through PDAs, wearables, and ubiquitous devices, computing technology is becoming more and more integrated with us and our everyday world. Unlike the stable static environment of desktop computers, which can rely on the user working within the virtual desktops that they present, these new forms of computing device will find themselves immersed in highly dynamic physical environments, in which the computer must take more responsibility in integrating itself into the user's world rather than vice versa. Context-awareness is a key enabling factor in supporting computers in these new environments, enabling them to sense their physical (and sometimes virtual) context and react and adapt to it in order to provide better, more appropriate, or completely new services to the user.

The ambition of this thesis is to demonstrate the new application possibilities which context-awareness presents and that by providing the necessary supporting architecture, user interface guidelines, and better understanding of context itself, will encourage, enhance and support the creation of context-aware software. The diagram in Figure 51, first presented in the chapter one, illustrates the areas of the context-aware research domain to which this thesis has contributed. The grey area with the dashed outline does not intend to show the *amount* of work in each area, but rather the *scope* of this work within the whole domain. For example, our context-aware infrastructure work is very broad in scope, aiming to support any context-aware software, whereas our applications research is tightly focused on fieldwork activities (although, of course, there are many lessons from this work that can be used in other types of application too).

9.1   Overview of Contribution

In the following sections we summarise our work in each of the context-aware software layers, as illustrated in the diagram below, describing the contribution that we have made and suggesting areas for future work.
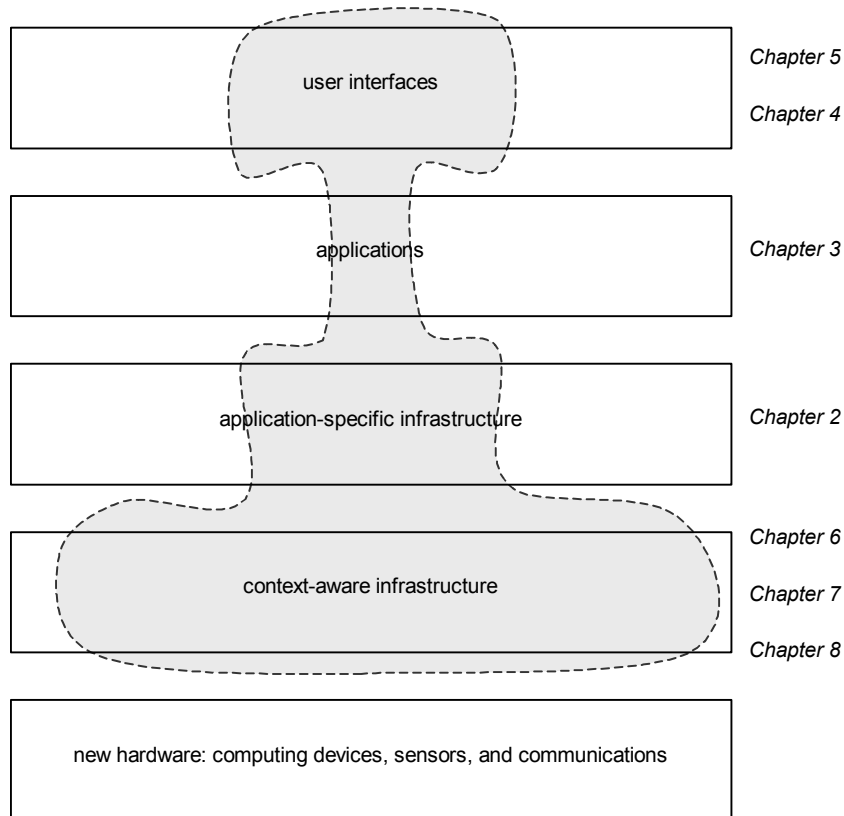
Figure 51. Illustration of the areas in which this thesis has contributed.

9.1.1   User Interfaces

In our user interface work we made a distinction between a mobile device that is used statically, and a mobile device that is used whilst mobile. The latter case of mobile usage was our main area of interest and was the focus of our investigations in the ecology fieldwork application domain.

We identified four attributes of mobile workers: "dynamic user configuration", "limited attention capacity", "high speed interaction" and "context dependency". We found context-awareness to be useful in helping computing devices better address some of these issues in their user interfaces. We also invented the MAUI, a user interface that

aims to reduce the attention required of the user, but not necessarily the number of interactions. This is achieved through an appropriate selection of modes of interaction for a particular activity. In an attempt to enable other developments to effectively reuse the principles that we have discovered, we devised a general-purpose system whereby the characteristics of the user's activity and of the different user interface mechanisms available could be plotted onto tables, from which their compatibility could be compared. A designer can build up a library of these tables to support the design of future user interfaces, helping achieve the 'minimal attention' in the 'minimal attention user interface' through the effective use of modes of interaction that least distract or interfere with those that the user is employing in her current activity.

9.1.1.1   Contribution

This user interface work contributes by providing a better understanding of the characteristics of the users and environments that new types of computing device intended for mobile usage will face. We have also created the novel concept of MAUIs, and have described and demonstrated how user interfaces can be created for mobile devices in order to best suit their user's activity. And through our method of identifying the preferences and restrictions in different modes of interaction for both activity and user interface controls, we have provided a concrete way in which designers can easily employ these design guidelines.

9.1.1.2   Future Work

Although we have presented two practical implementations of the MAUI concept in the user interface (a thumb-operated control for one-handed use, and a tactile control for eyes-free use), a great deal more work is required in creating new types of user interface controls to support different modes of interaction. Work is also required in investigating how these controls can be bound together to form coherent user interfaces. An important aspect, which was beyond the scope of our thesis research, is in combining the context-aware and MAUI techniques. Rather than creating static user interfaces, context-aware MAUIs could enable user interfaces to dynamically and autonomously change so as to continuously present themselves in the most suitable form for the user.

9.1.2   Applications

Our focus for applications research was in the domain of fieldwork, where we collaborated with a group of ecologists on a number of relatively large-scale experiments. We applied the concept of stick-e notes as a basis for some data collection fieldwork software we created, running on a PalmPilot device connected to a GPS receiver. This suite of three context-aware applications enabled an MSc ecology student to complete her summer of giraffe observation fieldwork in less time, and with more data collected, than would have been possible without the tools.

In addition to the generic data collection tools, we also developed a rhino identification tool in which context played an important role. A number of context elements, such as the location, footprint dimensions, substrate and footedness, were utilised in a program that aided ecologists in identifying the rhino whose footprint had been discovered.

Besides illustrating the exciting new possibilities of context-aware software for new computing devices, we also wished to use these applications to examine if they would be as useful as our (and others') work had suggested. To this end we conducted an extensive usability study with forty ecology fieldwork volunteers over a period of five months. These volunteers came from a wide variety of backgrounds, with a large percentage of them being novices to both computing and fieldwork. Their acceptance of our tools was unanimous with all participants preferring them to the manual paper-based solution, and with the majority of individuals managing to learn how to use them within half an hour, and feeling comfortable with the technology within their first day of fieldwork. The top four reasons given for the tools' popularity were faster data entry data (some fields, such as location, being automatically completed), an increase in efficiency with less work and writing, easier transcription of the data (automated download), and less chance for error (by automatic collection of complex data such as location, and through standardised forms with embedded input validity checks).

9.1.2.1   Contribution

Our work on applications has demonstrated how new types of practical context-aware computing tools can be employed in new application domains, which in this case had previously been without any form of computing support. The applications have shown how context-awareness can be used effectively, and through these applications we have

contributed to a better understanding of context-awareness, stick-e notes, and the surrounding processing and user interface issues. To establish whether these new types of computing technology would actually be useful to real users in their everyday work environments we conducted an extensive usability study. The result was resoundingly positive in that the acceptance by users was unanimous, and the application of context-awareness well received.

We hope that these applications will actively encourage future work on building new context-aware software in new application domains, fieldwork being just one such domain. We hope that the demonstrated user acceptance of the fieldwork tools will help to show that, rather than fanciful ideas, these new concepts and devices can be very successful and useful in helping user's with practical real-world activities.

9.1.2.2   Future Work

With regard to the fieldwork tools in particular, in future work we would like to build better user interfaces based on our HCI research and employ more types of sensor to enable better context-awareness. There is also the possibility for other types of application for the field that concentrate on presenting data to researchers and/or tourists, rather than in collecting and recording it. It is likely that such applications could make much fuller use of the triggering concept and perhaps incorporate wireless communications technologies in order to bring remote digital resources out into the appropriate context in the field.

9.1.3   Application Infrastructure

Our first foray into context-awareness was in the development of the stick-e note framework. Stick-e notes were based on Brown's concept of an electronic Post-It note [Brown 1996 - #15], allowing digital data to be attached to physical contexts just as how a paper-based note could be attached to a physical object using conventional Post-It notes. The stick-e note proved to be much more versatile than its manual counterpart, potentially allowing any form of digital data to be attached to any form of context or combination of contexts, e.g. location, temperature, time, etc.

We designed the stick-e note framework to provide a foundation for applications that need to work with electronic resources associated with physical contexts. Application processing revolved around three fundamental processes: attaching (where electronic data is attached to physical contexts), matching (where stored stick-e notes are compared to see if their context matches the current context), and dispatching (where any stick-e notes that matched are delivered to interested clients). The application data was stored as stick-e notes, consisting of context and content data.

9.1.3.1   Contribution

The stick-e note framework was our initial attempt at supporting context-aware application development through a generic storage and processing model (i.e. stick-e notes and the attaching, matching, and dispatching processes). Although not covering all types of context-aware applications, we provided a good model for those that work with data that is in some way associated with the physical world. A key part of this model is triggering, where electronic resources associated with a set of physical conditions are automatically delivered to interested clients when those conditions occur.

9.1.3.2   Future Work

We believe that the application semantics that the framework provides are very useful, but rather than providing all the services from capturing context to delivering stick-e notes, it should focus solely on the functions directly involving stick-e notes. In future work we would like to replace the indirect functions with the context information service that we summarise in the following section. We believe that in the future context-aware services will be developed as stacks, with basic context-capturing services at the bottom supporting more application-specific ones, such as stick-e notes, at the top.

More effective triggering engines is a specific area of concern for future work, as is the need to present triggering to the user in a more comprehensive and understandable way, e.g. explanations of what caused the triggering, providing "early warning systems" and not just a binary triggered event, and presenting (and working with) context data in more understandable forms than the raw data extracted from sensors. Of course, there

is also a need for work on complementary application frameworks to support other types of applications in which stick-e notes are not appropriate.

9.1.4   Context-Aware Infrastructure

After completing our work on context-aware applications we re-examined the nature of context and context-aware applications. Finding each to be more complex than we had supposed at the beginning of our research we were forced to re-evaluate the role of our supporting infrastructure, which at that point consisted solely of the stick-e note framework. We defined a lower-layer piece of supporting infrastructure which we called the "context information service". The responsibilities of this service are to gather, model, and provide contextual data, i.e. the general functionality that is required at the heart of all context-aware applications. It is the common base on top of which more sophisticated context-aware capabilities and applications may be built.

We performed a thorough examination of research in the area, selecting eleven infrastructure projects in particular with which to examine the current state of the art. We divided the functionality that we believed to be necessary of a context information service into five layers: sensor support, context extraction mechanisms, context data type support, context model, and client interface. We also considered the ability to distribute the service's sensors, context model, and processing. Examining each project layer by layer we found that most work had biases to particular types of application or context, and that none provided what we regarded as a complete solution. This state of affairs led us to propose our own solution for a context information service consisting of a set of service components and modelling components that can be used to represent and monitor the context of physical and virtual objects. We also outlined some initial ideas for distributing the CIS.

9.1.4.1   *Contribution*

We have contributed to the understanding of context and context-awareness, developing a more complete view of their often-complex nature than other models. To support context-aware applications in working with this context data we defined the context information service, which forms the core support in a context-aware software stack. We have specified the desirable characteristics of a CIS at five different levels of

functionality, enabling the analysis of context-aware infrastructures and identification of their strengths and weaknesses. Seeing the lack of a complete service in existing solutions we have designed our own context information service that meets all of the requirements and provides context-aware software developers with a simple yet very flexible context-aware infrastructure. Finally, we have also presented some novel concepts for the distribution of a CIS.

9.1.4.2   Future Work

We welcome further work on using our definition of a CIS to evaluate other services and to help extend them to form a complete CIS. Perhaps further research will also require additions and adaptations to the initial CIS requirements proposed here. In regard to our own CIS design, a full implementation must be created and a number of different client applications constructed in order to evaluate it conclusively, as is the case for our suggested methods of distribution. Our proposals for distribution of the CIS revolved around the world service component; further work is needed on how to distribute the other service components.

9.2   Research Aims Revisited

Throughout this thesis we have explored context-awareness and expanded the understanding of both context and context-aware applications. We have investigated the need for infrastructures to support the development of context-aware applications and have developed both an application-oriented framework and a context information service, together these form the basis of a context-aware software stack. Through the development of some specialist applications for fieldworkers we have demonstrated how context-awareness can be effectively used to create compelling applications for new types of computing technology in new application domains, and effectively shown that people are eager to use them. In our user interface research we have presented some guidelines for developing new types of user interface for these applications, and also presented some new interface controls for eyes-free and one-handed use.

Our research has examined a slice of context-aware computing software spanning user interface to underlying infrastructure, and has aimed to contribute at each level. We believe that our infrastructure level research will lead to a better understanding of

context and the means with which context-awareness can be more widely and easily utilised. Our applications research demonstrates how these new technologies can be used effectively, and how whole new application domains can be discovered. Finally our user interface research provides the necessary guidelines to ensure that these new applications will integrate with the world of the user and prove effective and convenient tools.

In the introduction we defined the aims of the thesis in terms of three levels. We conclude by revisiting each of these levels here and comment on our contribution:

Level 1: Infrastructure

Research Aim: to investigate if software infrastructure to support context-awareness is required, and if so, to determine what services that infrastructure should provide and how it should provide them.

Contribution: we developed a prototype application framework called stick-e notes (presented in chapter two) and a number of context-aware applications (presented in chapters two and three). These demonstrated the complex nature of context and certainly showed the need for supporting infrastructure. We further used our applications and framework research, and that of other researchers, to support the development of a general set of requirements for a context-aware infrastructure, as described in chapter six. Finally, a prototype context information service was proposed to illustrate how such an infrastructure could be realised and distributed, presented in chapters seven and eight.

Level 2: Applications

Research Aim: to further the understanding of context and context-awareness through empirical experiments, to explore practical applications of context-awareness, and to find out if context-awareness can be usefully employed in practice.

Contribution: all of our research on stick-e note frameworks (presented in chapter two), electronic Post-It applications (also presented in chapter two), and data

collection applications (presented in chapter three) helped to promote a further understanding of context and how it can be used within applications. In chapter six we condensed our findings from the different applications into a set of characteristics defining context and a set of requirements of context-aware applications. Finally, to ascertain a tangible measure of how useful context-awareness is in practice, we performed an extensive field trial and usability study (detailed in chapter five), which showed a very positive response towards our context-aware applications (and the context-aware features in particular).

Level 3: HCI

Research Aim: to explore new methods of human-computer interaction for this new area of computing, to determine their usability, and to establish some general HCI design principles.

Contribution: using our prototype applications as a foundation, in chapter five we explored the unique environment and characteristics of our users and developed the concept of the Minimal Attention User Interface. This concept emphasises the reduction of attention necessary by the user but not necessarily the amount of interaction. We created some new hands-free and eyes-free data input methods to put our ideas to practical use, and developed a general set of MAUI design guidelines to aid in realising this concept in other applications.

In summary, we believe that our work has demonstrated the new application possibilities which context-awareness presents and has defined the necessary supporting software infrastructure, user interface guidelines, and better understanding of context itself, that will encourage and enable the creation of context-aware software.

Table of References

1. Abowd, G., CyberGuide: A Mobile Context-Aware Tour Guide, *Wireless Networks*, 1997(3), pp. 421-433.

2. Abowd, G.D., Dey, A., and Wood, A., Applying Dynamic Integration as a Software Infrastructure for Context-Aware Computing, *Georgia Tech*, Atlanta, 1997, GIT-GVU-97-18.

3. Addlesee, M., ORL Active Floor, *IEEE Personal Communications*, 1997, 4(5), pp. 35-41.

4. Adly, N., Steggles, P., and Harter, A., SPIRIT: a Resource Database for Mobile Users, *ACM CHI'97 Workshop on Ubiquitous Computing*, March 1997, Atlanta, Georgia, USA.

5. Asthana, A., Cravatts, M., and Krzyanowski, An Indoor Wireless System for Personalised Shopping Assistance, *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, December 8-9 1994, New Orleans, USA, pp. 69-74.

6. Bailey, K., Stone Age Skills Meet the Challenge of the Techno-Age, *The Times,* 8th October 1997, London, pp. 16-17 (Interface section).

7. Bartlett, J., F, W4 - The Wireless World Wide Web, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, pp. 176-178.

8. Bennet, F. and Harper, A., Low Bandwidth Infra-Red Networks and Protocols for Mobile Communicating Devices, *Olivetti*, 1993.

9. Bennett, F., Richardson, T., and Harter, A., Teleporting - Making Applications Mobile, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, pp. 82-84.

10. Bennett, P., Peter Bennett's GPS and NMEA Site, http://vancouver-webpages.com/peter/idx_nmeadoc.html, *Peter Bennett,* 31/07/2001.

11. Bluetooth-Consortium, Bluetooth Homepage, http://www.bluetooth.com/, *Bluetooth-Consortium,* 01/01/2001.

12. Borovoy, R., McDonald, M., Martin, F., and Resnick, M., Things that Blink: Computationally Augmented Name Tags, *IBM Systems Journal*, 1996, 35(3&4), pp. 488-495.

13. Bowskill, J., Morphett, J., and Downie, J., A Taxonomy for Enhanced Reality Systems, *1st International Symposium on Wearable Computers*, 13-14 October 1997, Cambridge, MA, USA, pp. 175-176.

14. Brody, A.B. and Gottsman, E.J., Pocket BargainFinder: A Handheld Device for Augmented Commerce, *First International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, September 1999, Karlsruhe, Germany, pp. 44-51.

15.   Brown, P.J., The Stick-e Document: a Framework for Creating Context-Aware Applications, *EP '96*, September 1996, Palo Alto, CA, USA, pp. 259-272.

16.   Brown, P.J., Bovey, J.D., and Chen, X., Context-Aware Applications: From the Laboratory to the Marketplace, *IEEE Personal Communications*, 1995, 4(5), pp. 58-64.

17.   Brown, P.J. and Jones, G.J.F., Context-Aware Retrieval: Exploring a New Environment for Information Retrieval and Information Filtering, *Personal & Ubiquitous Computing*, 2001, 5(4), pp.253-263.

17a.   Brown, P.J, Burleson, W., Lamming, M., Rahlff, O-W., Romano, G., Scholz, J., and Snowdon, Context-awareness: some   compelling   applications, http://www.dcs.ex.ac.uk/~pjbrown/papers/compelling.html, 31/07/2001.

18.   Bruegge, B. and Bennington, B., Applications of Mobile Computing and Communication, *IEEE Personal Communications*, 1996(February), pp. 64-71.

19.   Butz, A., Hollerer, T., Feiner, S., McIntyre, B., and Beshers, C., Enveloping Users and Computers in a Collaborative 3D Augmented Reality, *IWAR '99*, October 1900, San Francisco, CA, USA, pp. 35-44.

20.   Chavez, E., Ide, R., and Kirste, T., SAMoA: An Experimental Platform for Situation-Aware Mobile Assistance, *IMC '98*, November 1998, Rostock, Germany.

21.   Cheverst, K., Davies, N., Mitchell, K., and Friday, A., Experiences of Developing and Deploying a Context Aware Tourist Guide: The GUIDE Project, *MOBICOM 2000*, August 2000, Boston, MA, USA, pp. 20-31.

22.   Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C., Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences, *CHI 2000*, April 2000, Netherlands, pp. 17-24.

23.   CORBA, CORBA Homepage, http://www.corba.org/, *CORBA,* 31/07/2001.

24.   Davies, N., Cheverst, K., Mitchell, K., and Friday, A., Caches in the Air: Disseminating Information in the Guide System, *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, February 1999, New Orleans, USA, pp. 11-19.

25.   Davies, N., Friday, A., Wade, S., and Blair, G., L^2imbo: A Distributed Systems Platform for Mobile Computing, *ACM Mobile Networks and Applications (MONET)*, 1998, 3(2), pp. 143-156.

26.   Davies, N., Wade, S., Friday, A., and Blair, G., Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications, *International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP'97)*, May 1997, Toronto, Canada, pp. 291-302.

27.   Demers, A.J., Research Issues in Ubiquitous Computing, *13th Annual ACM Symposium on Principles of Distributed Computing*, 14-17 August 1994, Los Angeles, CA, USA, pp. 2-9.

28. Dey, A. and Abowd, G., The Context Toolkit: Aiding the Development of Context-Aware Applications, *Workshop on Software Engineering for Wearable and Pervasive Computing (SEWPC)*, June 2000, Limerick, Ireland.

29. Dey, A., Abowd, G., and Salber, D., A Context-based Infrastructure for Smart Environments, *MANSE'99*, December 1999, pp. 114-128.

30. Dey, A., Salber, D., and Abowd, G., The Conference Assistant: Combining Context-Awareness with Wearable Computing, *ISWC '99*, October 1999, San Francisco, CA, USA, pp. 21-28.

31. Dey, A.K. and Abowd, G.D., CyberDesk: The Use of Perception in Context-Aware Computing, *PUI 97 Perceptual User Interfaces Workshop*, October 1997, pp. 26-27.

32. Dey, A.K. and Abowd, G.D., CybreMinder: a context-aware-system for supporting reminders, *Personal Technologies*, 2000, 4(4), pp. 172-186.

33. Dias, A.E., Santos, E.M., Pimentão, J.P., Pedrosa, P., and Romão, T.I., Bringing Ubiquitous Computing Into Geo-Referenced Information Systems, *Second Joint European Conference & Exhibition on Geographical Information*, Barcelona, Spain, pp. 100-103.

34. Dix, A., Co-operation Without (reliable) Communication: Interfaces for Mobile Applications, *Distributed Systems Engineering*, 2(3), pp. 171-181.

35. Duffet-Smith, P., High Precision CURSOR and Digital CURSOR: the Real Alternatives to GPS and Control, *EURONAV 96 Conference on Vehicle Navigation*.

36. Elrod, S., Hall, G., Costranza, R., Dixon, M., and Des Rivieres, J., Responsive Office Environments, *Communications of the ACM*, 1993, 36(7), pp. 84-85.

37. Emmerson, B., The Mobile Intranet, *Byte*, 1998(May 1998), pp. 15-19.

38. Fadel, C., PDAs Will Come of Age in the Nineties, *Electronic Design*, 1995, 43(1), pp. 104-106.

39. FCC, Revision of the Commission's Rules To Ensure Compatibility with Enhanced 911 Emergency Calling Systems (Wireless E911 Rules), *Federal Communications Commission*, 1996, OGC-96-34.

40. Feiner, S., The Importance of Being Mobile: Some Social Consequences of Wearable Augmented Reality Systems, *IWAR'99*, October 1999, San Francisco, CA, USA, pp. 145-148.

41. Feiner, S., MacIntyre, B., Haupt, M., and Solomon, E., Windows on the World: 2D Windows for 3D Augmented Reality, *UIST '93*, November 1993, pp. 145-155.

42. Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A., A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment, *1st Wearable Symposium*, 13-14 October 1997, Cambridge, MA, USA, pp. 74-81.

43. Feiner, S., MacIntyre, B., and Seligmann, D., Knowledge-based Augmented Reality, *Communications of the ACM*, 1993, 36(7), pp. 52-62.

44. Fickas, S., Kortuem, G., and Segall, Z., Software Organisation for Dynamic and Adaptable Wearable Systems, *ISWC '97*, October 1997, Cambridge, MA, USA, pp 56-64.

45. Finger, S., Terk, M., Subrahmanian, E., Kasabach, C., Prinz, F., Siewiorek, D., Smailagic, A., Stivoric, J., and Weiss, L., Rapid Design and Manufacture of Wearable Computers, *Communications of the ACM*, 1996, 39(2), pp. 63-70.

46. Fitzmaurice, Situated Information Spaces and Spatially Aware Palmtop Computers, *Communications of the ACM*, 1993, 36(7), pp. 38-49.

47. Fitzmaurice, G. and Buxton, W., The Chameleon: Spatially Aware Palmtop Computers, *CHI '94*, April 1994, Boston, MA, USA, pp. 451-452.

48. Fitzmaurice, G.W., Zhai, S., and Chignell, M.H., Virtual Reality for Palmtop Computers, *ACM Transactions on Information Systems*, 1993, 11(3), pp. 197-218.

49. Fletcher, R., Force Transduction for Human-Technology Interfaces, *IBM Systems Journal*, 1996, 35(3&4), pp. 630-638.

50. Foner, L.N., Artificial Synesthesia via Sonification: A Wearable Augmented Sensory System, *ISWC'97*, October 1997, Cambridge, MA, USA, pp. 156-157.

51. Foster, C.D., PDAs and the Library Without a Roof, *Journal of Computing in Higher Education*, 1995, 7(1), pp. 85-93.

52. Freedman, N. and Pollock, I., Ward Round - Mobile Computing System Trial, *Current Perspectives in Healthcare Computing Conference*, 18-20 March 1996, pp. 661-671.

53. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns, 1st Edition, 1995, Addison-Wesley, 0201633612.

54. Gellersen, H.-W., Beigl, M., and Schmidt, A., Environment-Mediated Mobile Computing, *Symposium on Applied Computing '97*, February-March 1999, San Antonio, TX, USA, pp. 416-418.

55. Gessler, S. and Kotulla, A., PDAs as Mobile WWW Browsers, *Computer Networks and ISDN Systems*, 1995(28), pp. 53-59.

56. Goldberg, L., Wireless LAN's: Mobile-Computing's Second Wave, *Electronic Design*, 1995, 43(13), pp. 55-72.

57. Harrison, B.L., Fishkin, K.P., Gujar, A., Mochon, C., and Want, R., Squeeze Me, Hold Me, Tilt Me! An Exploration of Manipulative User Interfaces, *CHI'98*, April 1998, Los Angeles, CA, USA, pp. 17-24.

58. Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P., The Anatomy of a Context-Aware Application, *5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 1999)*, August 1999, Seattle, WA, USA, pp. 59-68.

59. Herring, The Global Positioning System, *Scientific American*, 1996(February), pp. 44-50.

60.   Herzberg, A., Krawczyk, H., and Tsudik, G., On Travelling Incognito, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, pp. 205-211.

61.   Hindus, D., Arons, B., Stifelman, L., Gaver, B., Mynatt, E., and Back, M., Designing Auditory Interactions for PDAs, *UIST 95*, November 14-17 1995, Pittsburgh, PA, USA, pp. 143-146.

62.   Hollerer, T. and Pavlik, J., Situated Documentaries: Embedding Multimedia Presentations in the Real World, *ISWC '99*, October 1999, San Francisco, CA, USA, pp. 79-86.

63.   Honeyman, P. and B, H.L., Communications and Consistency in Mobile File Systems, *IEEE Personal Communications*, 1995, 2(6), pp. 44-48.

64.   Hull, R., Neaves, P., and Bedford-Roberts, J., Towards Situated Computing, *1st International Symposium on Wearable Computers (ISWC '97)*, 13-14 October 1997, Cambridge, MA, USA, pp. 146-153.

65.   Imielinski, T. and Badrinath, B.R., Mobile Wireless Computing, *Communications of the ACM*, 1994, 37(19), pp. 18-28.

66.   Infrared-Data-Association, IrDA Homepage, http://www.irda.org/, *Infrared Data Association (IrDA)*, 31/07/2001.

67.   Jameel, A., Stuempfle, M., Jiang, D., and Fuchs, A., Web on Wheels: Toward Internet-Enabled Cars, *IEEE Computer*, 1998(January 1998), pp. 69-76.

68.   Janin, A., Zikan, K., Mizell, D., Banner, M., and Sowizral, H., A Videometric Head Tracker for Augmented Reality Applications, *SPIE*, pp. 308-315.

69.   Kaashoek, F., Pinckney, T., and Tauber, J.A., Dynamic Documents: Mobile Wireless Access to the WWW, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, pp. 179-184.

70.   Kalakota, R., Stallaert, J., and Whinston, A.B., Mobile Agents and Mobile Workers, *29th Hawaii International Conference on System Sciences*, pp. 354-365.

71.   Kamba, T., Elson, S., Harpold, T., Stamper, T., and Sukaviriya, P., Using Small Screen Space More Efficiently, *Human Factors in Computing Systems*, April 1996, Vancouver, Canada, pp. 383-390.

72.   Kirch, D., Starner, T., and Assefa, S., The Locust Swarm: An Environmentally-powered, Networkless Location and Messaging System, *ISWC '97*, October 1997, Cambridge, MA, USA, pp. 169-170.

73.   Kortuem, G., Segall, Z., and Bauer, M., Context-Aware, Adaptive Wearable Computers as Remote Interfaces to 'Intelligent' Environments, *Second International Symposium on Wearable Computers (ISWC'98)*, October 1998, Pittsburgh ,PA, USA, pp. 58-65.

74.   Kristoffersen, S. and Ljungberg, F., "Making Place" to Make IT Work: Empirical Explorations of HCI for Mobile CSCW, *International ACM SIGGROUP conference on Supporting Group Work*, November 1999, Phoenix, AZ, USA, pp. 276-285.

75.   Kristoffersen, S. and Ljungberg, F., Mobile Use of IT, *IRIS22*, August 1999, Jyvaskyla, Finland, Vol. 2, pp. 271-284.

76.   Lamming, M., Brown, P., Carter, K., Eldridge, M., Flynn, M., Louie, G., Robinson, P., and Sellen, A., The Design of a Human Memory Prosthesis, *The Computer Journal*, 1994, 37(4), pp. 153-163.

77.   Lamming, M. and Flynn, M., "Forget-me-not" Intimate Computing in Support of Human Memory, *EuroPARC*, Cambridge, UK, 1994, EPC-1994-103.

78.   Leonhardt, U. and Magee, J., Multi-Sensor Location Tracking, *ACM/IEEE International Conference on Mobile Computing and Networking*, October 1998, Dallas, TX, USA, pp. 203-214.

79.   Leonhardt, U. and Magee, J., Security Considerations for a Distributed Location Service, *Journal of Network and Systems Management*, March 1998, 6(1), pp. 51-70.

80.   Leonhardt, U. and Magee, J., Towards a General Location Service for Mobile Environments, *Third IEEE Workshop on Services in Distributed and Networked Environments*, June 1996, Macau, pp. 43-50.

81.   Leonhardt, U., Magee, J., and Dias, P., Location Service in Mobile Computing Environments, *Computer & Graphics*, 1996, 20(5), pp. 627 632.

82.   lesswire, lesswire AG's homepage, http://www.lesswire.de/, *lesswire AG,* 31/07/2001.

83.   Lewis, T., Information Appliances: Gadget Netopia, *IEEE Computer*, 1998(January), pp. 59-68.

84.   Long, S., Aust, D., Abowd, G., and Atkeson, C., Cyberguide: Prototyping Context-Aware Mobile Applications, *ACM CHI 96 Conference on Human Factors in Computing Systems*, pp. 293-294.

85.   Long, S., Kooper, R., Abowd, G.D., and Atkeson, C.G., Rapid Prototyping of Mobile Context-Aware Applications: The Cyberguide Case Study, *2nd ACM International Conference on Mobile Computing and Networking (MobiCom '96)*, November 1996, pp. 97-107.

86.   Maaß, H., Location-aware Mobile Applications based on Directory Services, *Mobile Networks and Applications*, 1998, 3(2), pp. 157-173.

87.   Mann, S., Smart Clothing: The Shift to Wearable Computing, *Communications of the ACM*, 1996, 39(8), pp. 25-26.

88.   Mann, S., Wearable Computing: A First Step Toward Personal Imaging, *Computer*, 1997(February 1997), pp. 25-32.

89.   Meth, C., PDAs: What will it Take to Satisfy Users?, *Electronic Design*, 1994, 42(26), pp. 49-54.

90.   MIT, Things That Think Homepage, http://www.media.mit.edu/ttt/, *MIT,* 31/07/2001.

91. Mukherjee, A. and Siewiorek, D., Mobility: A Medium for Computation, Communication, and Control, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, Santa Cruz, CA, USA, pp. 8-11.

92. NMEA, NMEA 0183 Info, http://www.nmea.org/0183.htm, *National Marine Electronics Association*, 31/07/2001.

93. Noble, B., Price, M., and Satyanarayana, M., A Programming Interface for Application-Aware Adaptation in Mobile Computing, *Computing Systems*, 1995, 8(4), pp. 345-363.

94. Not, E., Petrelli, D., Stock, O., Strapparava, C., and Zancanaro, M., Person-Oriented Guided Visits in a Physical Museum, *Fourth International Conference on Hypermedia and Interactivity in Museums (ICHIM 97)*, September 1997, Paris, France, pp. 162-172.

95. Not, E. and Zancanaro, M., Content Adaptation for Audio-Based Hypertexts in Physical Environments, *Hypertext '98*, June 1998, Pittsburgh, PA, USA, pp. 25-31.

96. Palm, Palm Computing Homepage, http://www.palm.com/, *Palm Computing Inc.*, 31/07/2001.

97. Paradiso, J.A., The Interactive Balloon: Sensing, Actuation, and Behaviour in a Common Object, *IBM Systems Journal*, 1996, 35(3&4), pp. 473-488.

98. Pascoe, J., Adding Generic Contextual Capabilities to Wearable Computers, *The Second International Symposium on Wearable Computers*, 19-20 October 1998, Pittsburgh, PA, USA, pp. 92-99.

99. Pascoe, J., Morse, D.R., and Ryan, N.S., Developing Personal Technology for the Field, *Personal Technologies*, 1998, 2(1), pp. 28-36.

100. Pascoe, J. and Ryan, N., Mobile Computing in Fieldwork Environments Homepage, http://www.cs.ukc.ac.uk/projects/mobicomp/Fieldwork/index.html, *Computing Lab, University of Kent at Canterbury*, 01/01/2000.

101. Pascoe, J., Ryan, N., and Brown, P., Context Aware: the Dawn of Sentient Computing?, *GPS World*, 1998, 9(9), pp. 22-30.

102. Pascoe, J., Ryan, N., and Morse, D., Issues in Developing Context-Aware Computing Applications, *HUC '99*, September 1999, Karlsruhe, Germany, pp. 208-221.

103. Pascoe, J., Ryan, N.S., and Morse, D.R., Human Computer Giraffe Interaction: HCI in the Field, *Workshop on Human Computer Interaction with Mobile Devices*, 21-23 May 1998, University of Glasgow, UK, GIST Technical Report G98-1.

104. Pedersen, E.R. and Sokolar, T., AROMA: Abstract Representation of Presence Supporting Mutual Awareness, *CHI '97*, March 1997, Atlanta, USA, pp. 51-58.

105. Pentland, A.P., Petrazzuoli, M., Gerega, A., and Starner, T., The Digital Doctor: An Experiment in Wearable Telemedicine, *The First International Symposium on Wearable Computers ISWC '97*, October 1997, Cambridge, MA, USA, pp. 173-174.

106.   Pfitzmann, A., Pfitzmann, B., and Waidner, M., Trusting Mobile User Devices and Security Modules*, IEEE Computer*, 1997(February 1997), pp. 61-68.

107.   Priyantha, N., Chakraborty, A., and Balakrishnan, H., The Cricket Location-Support System*, MOBICOM 2000*, August 2000, Boston, MA, USA, pp. 32-43.

108.   Randell, C. and Muller, H., The Shopping Jacket: Wearable Computing for the Consumer*, Personal Technologies*, 2000, 4(4), pp. 241-244.

109.   Rekimoto, J., Tilting Operations for Small Screen User Interfaces*, Symposium on User Interface Software and Technology (UIST '96)*, November 1996, Seattle, WA, USA, pp. 167-168.

110.   Rekimoto, J. and Ayatsuka, Y., CyberCode: Designing Augmented Reality Environments with Visual Tags*, Designing Augmented Reality Environments (DARE 2000)*, April 2000, Denmark, pp 1-10.

111.   Rekimoto, J., Ayatsuka, Y., and Hayashi, K., Augment-able Reality: Situated Communication through Digital and Physical Spaces*, ISWC '98*, November 1998, Pittsburgh, PA, USA, pp. 68-75.

112.   Rekimoto, J. and Nagao, K., The World through the Computer: Computer Augmented Interaction with Real World Environments*, UIST 95*, November 14-17 1995, Pittsburgh, PA, USA, pp. 29-36.

113.   Resnick, M., Things to Think With*, IBM Systems Journal*, 1996, 35(3&4), pp. 441-442.

114.   Resnick, M., Martin, F., Sargent, R., and Silverman, B., Programmable Bricks: Toys to Think With*, IBM Systems Journal*, 1996, 35(3&4), pp. 443-452.

115.   Rhodes, B., Margin Notes: Building a Contextually Aware Associative Memory*, IUI '00*, January 2000, New Orleans, LA, USA, pp. 219-224.

116.   Rhodes, B.J., The Wearable Rememberence Agent: A System for Augmented Memory*, Personal Technologies Special Issue on Wearable Computing*, 1997, 1, pp. 218-224.

117.   Rhodes, B.J. and Starnar, 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*, 1996, pp. 487-495.

118.   Rudall, B.H., Contemporary Systems and Cybernetics*, Kybernetes*, 1997, 26(4), pp. 370-388.

119.   Ryan, N.S., Pascoe, J., and Morse, D.R., Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*, Computer Applications in Archaeology 1997*, October 1997, Oxford, England.

120.   Salber, D., Dey, A., and Abowd, G., The Context Toolkit: Aiding the Development of Context-Enabled Applications*, CHI'99*, May 1999, Pittsburgh, PA, USA, pp. 434-441.

121.   Saldana, J. and Cohn, D.L., A Hybrid Model for Mobile File Systems*, IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, Santa Cruz, CA, USA, pp. 20-23.

122.   Sarkar, M., Snibbe, S.S., Tversky, O.J., and Reiss, S.P., Stretching the Rubber Sheet: a Metaphor for Viewing Large Layouts on Small Screens, *Symposium on User Interface Software and Technology (UIST'93)*, November 1993, Atlanta, GA, USA, pp. 81-91.

123.   Satyanarayanan, M., Mobile Information Access, *IEEE Personal Communications*, 1996(February), pp. 26-33.

124.   Satyanarayanan, M., Noble, B., Kumar, P., and Price, M., Application-Aware Adaptation for Mobile Computing, *Operating Systems Review.*, 1994, 29(1), pp. 52-55.

125.   Schilit, B., Adams, N., and Want, R., Context-Aware Computing Applications, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, Santa Cruz, CA, USA, pp. 85-90.

126.   Schilit, B.N., Douglis, F., Kristol, D.M., Krzyzanowski, P., Sienicki, J., and Trotter, J.A., TeleWeb: Loosely Connected Access to the World Wide Web, *Computer Networks and ISDN Systems*, 1996, 1996(28), pp. 1431-1444.

127.   Schilit, W.N., System Architecture for Context-Aware Mobile Computing, *Columbia University*, 1995.

128.   Schmidt, A., Beigl, M., and Gellersen, H.-W., There is More to Context than Location, *IMC '98*, November 1998, Rostock, Germany.

129.   Scholtz, J.C., Lockhart, P., Salvador, T., and Newbery, J., Design: no job too small, *CHI '97*, March 1997, Atlanta, GA, USA, pp. 447-454.

130.   Selker, T., New Paradigms for Using Computers, *Communications of the ACM*, 1996, 39(8), pp. 60-69.

131.   Slowman, M., Distributed Systems and Computer Networks, 1987, Prentice Hall, 0-13-215864-7.

132.   Smailagic, A. and Siewiorek, D.P., Modalities of Interaction with CMU Wearable Computers, *IEEE Personal Communications*, 1996, 1996(February), pp. 14-25.

133.   Smith, D.G., Personal Intelligent Communications, *BT Technology Journal*, 1995, 13(2), pp. 106-113.

134.   Starner, T., Mann, S., Rhodes, B., and Levine, J., Augmented Reality through Wearable Computing, *Presence*, 1997, 6(4), pp. 386-398.

135.   Starner, T., Rhodes, B., Weaver, J., and Pentland, A., Everyday-use Wearable Computers. June 15, 1999. (http://www.gvu.gatech.edu/ccg/publications/starner-everyday-use-061599.pdf)

136.   Stilp, L., Carrier and End-User Applications for Wireless Location Systems, *SPIE*, pp. 119-126.

137.   Strothotte, T., Fritz, S., Michel, R., Raab, A., Petrie, H., Johnson, V., Reichert, L., and Schalt, A., Development of Dialogue Systems for a Mobility Aid for Blind People: Initial Design and

Usability Testing, *ACM SIGCAPH Conference on Assistive Technologies*, April 1996, Vancouver, Canada, pp. 139-144.

138.   Sun, Java Homepage, http://www.javasoft.com/, *Sun Microsystems Inc,* 31/07/2001.

139.   Symbian, Symbian Homepage, *Symbian,* 31/07/2001.

140.   Tanenbaum, Distributed Operating Systems, First Edition, 1994, Prentice Hall, 0132199084.

141.   Texas-University,   Global   Positioning   System   Overview, http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html, *Department of Geography, University of Texas at Austin,* 31/07/2001.

142.   Thompson, S., Sheat, D., and Holland, R., Using Telecommunications to Deliver Real-Time Transport Information; 'Superoute 66 Live' Trial Results, *6th World Congress on Intelligent Transport Systems*, November 1999, Newcastle upon Tyne, UK.

143.   Verplaetese, P., Inertial Proprioceptive Devices: Self-Motion-Sensing Toys and Tools, *IBM Systems Journal*, 1996, 35(3&4), pp. 639-650.

144.   Voelker, G.M. and Bershad, B.N., Mobisaic: An Information System for a Mobile Wireless Computing Environment, *IEEE Workshop on Mobile Computing Systems and Applications*, December 8-9 1994, pp. 185-190.

145.   Want, R., Hopper, A., Falcao, V., and Gibbons, J., The Active Badge Location System, *ACM Transactions on Information Systems*, 1992, 10(1), pp. 91-102.

146.   Want, R., Shilit, B.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J., and Weiser, M., An Overview of the PARCTAB Ubiquitous Computing Experiment., *IEEE Personal Communications*, 1995, 2(6), pp. 28-43.

147.   Ward, A., Jones, A., and Hopper, A., A New Location Technique for the Active Office, *IEEE Personal Communications*, 1997, 4(5), pp. 42-47.

148.   Webster, A., Feiner, S., MacIntyre, B., Massie, W., and Krueger, T., Augmented Reality in Architectural Construction, Inspection, and Renovation, *ASCE Third Congress on Computing in Civil Engineering*, June 1996, Anaheim, CA, USA, pp. 913-919.

149.   Weiser, M., The Computer for the 21st Century, in *Scientific American*, September 1991, pp. 94-104.

150.   Weiser, M., The Future of Ubiquitous Computing, *Communications of the ACM*, 1998, 41(1), pp. 41-42.

151.   Welling, G. and Badrinath, B.R., A Framework for Environment Aware Mobile Applications, *17th International Conference on Distributed Computing Systems (ICDCS '97)*, May 1997, Maryland, USA, pp. 384-391.

152.   Wood, K.R., Richardson, T., Bennett, F., Harter, A., and Hopper, A., Global Teleporting with Java: Toward Ubiquitous Personalized Computing*, IEEE Computer*, 1997(February 1997), pp. 53-59.

153.   Wright, E., GPS Takes a Vacation, *GPS World*, June 1995, pp. 22-30.

154   Zimmerman, T.G., Personal Area Networks: Near-Field Intrabody Communications*, IBM Systems Journal*, 1996, 35(3&4), pp. 609-617.

Appendix A:

Stick-e Note Framework

Design Diagrams

Design diagrams are presented in Booch notation ["Object Oriented Analysis and Design with Applications", Grady Booch, 1993, ISBN 0805353402]
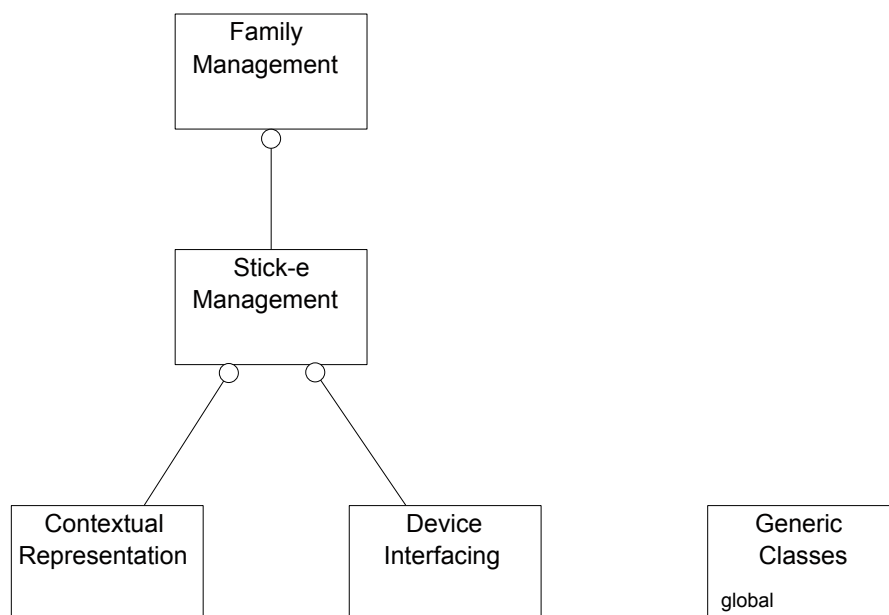
Figure 52. Top level class categories.
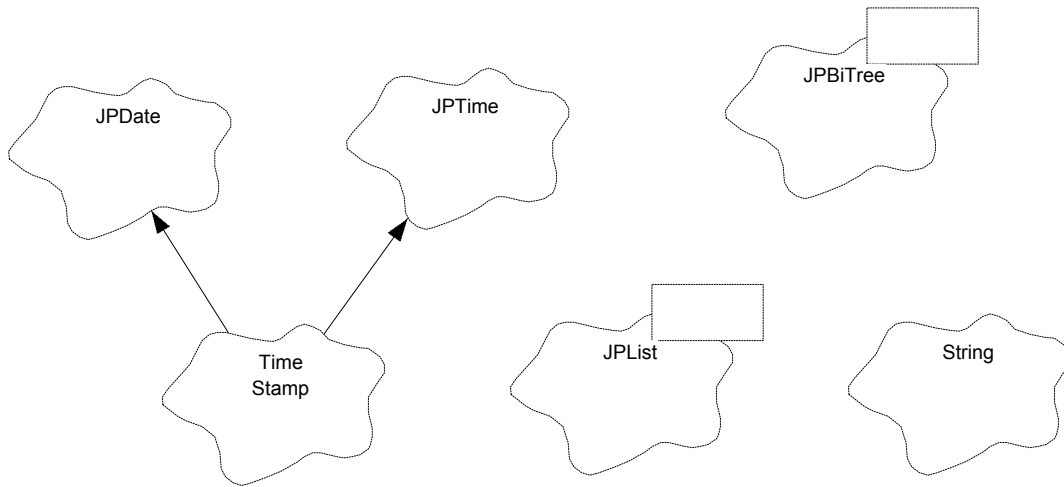
Figure 53. Class diagram for generic classes category.
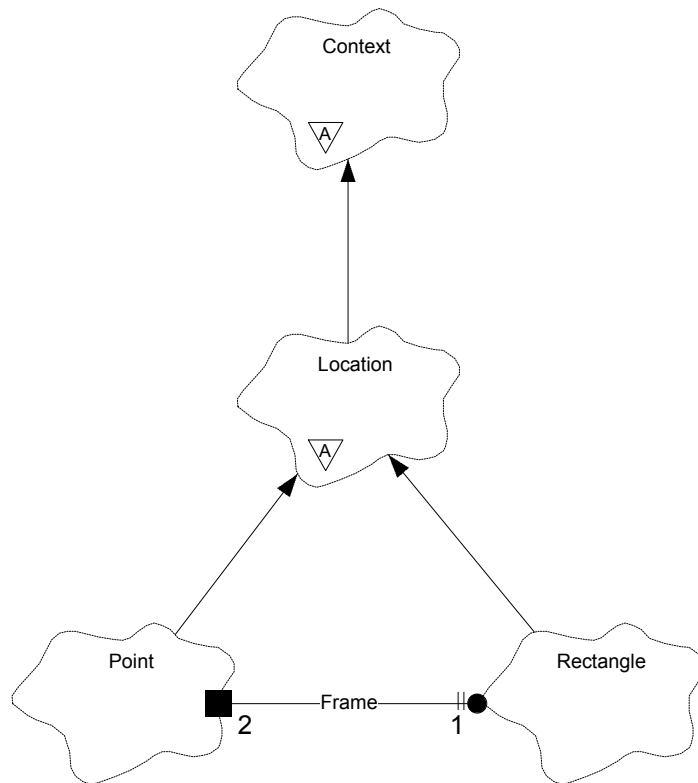
Figure 54. Class diagram for contextual representation category.
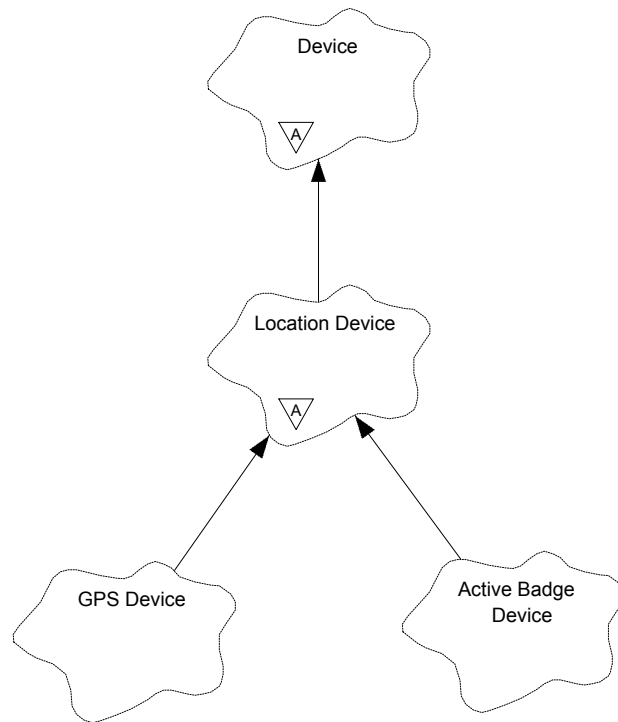
Figure 55. Class diagram for device interfacing category.
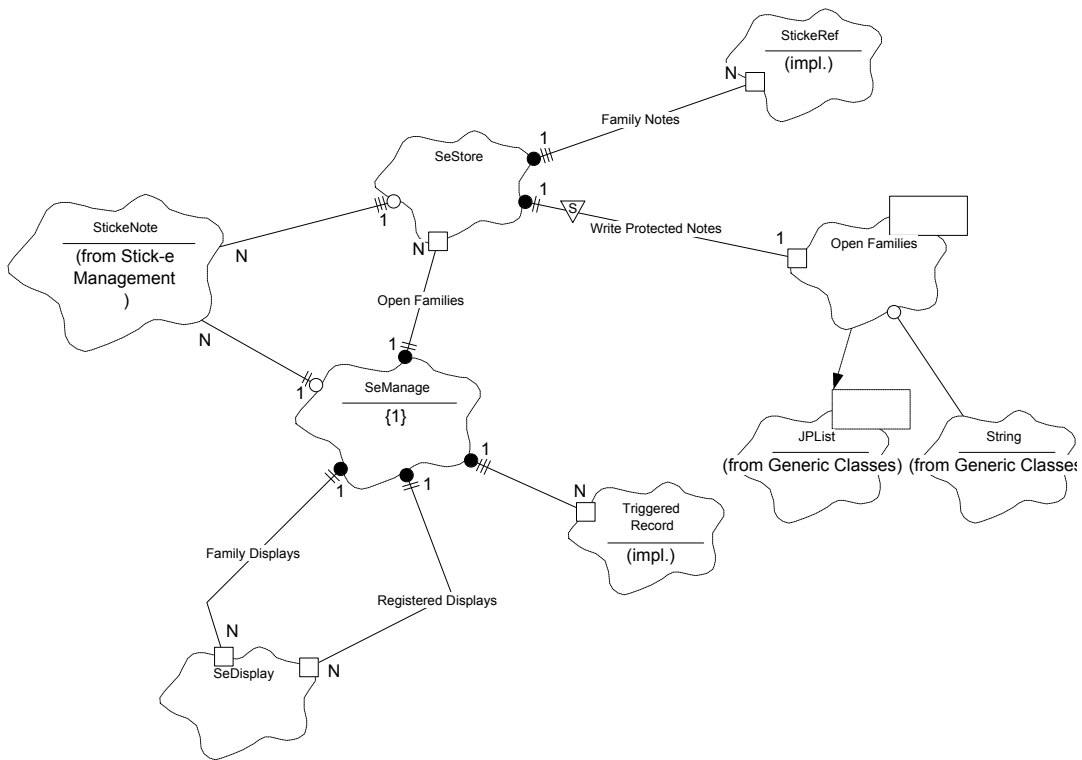
Figure 56. Class diagram for the family management category.
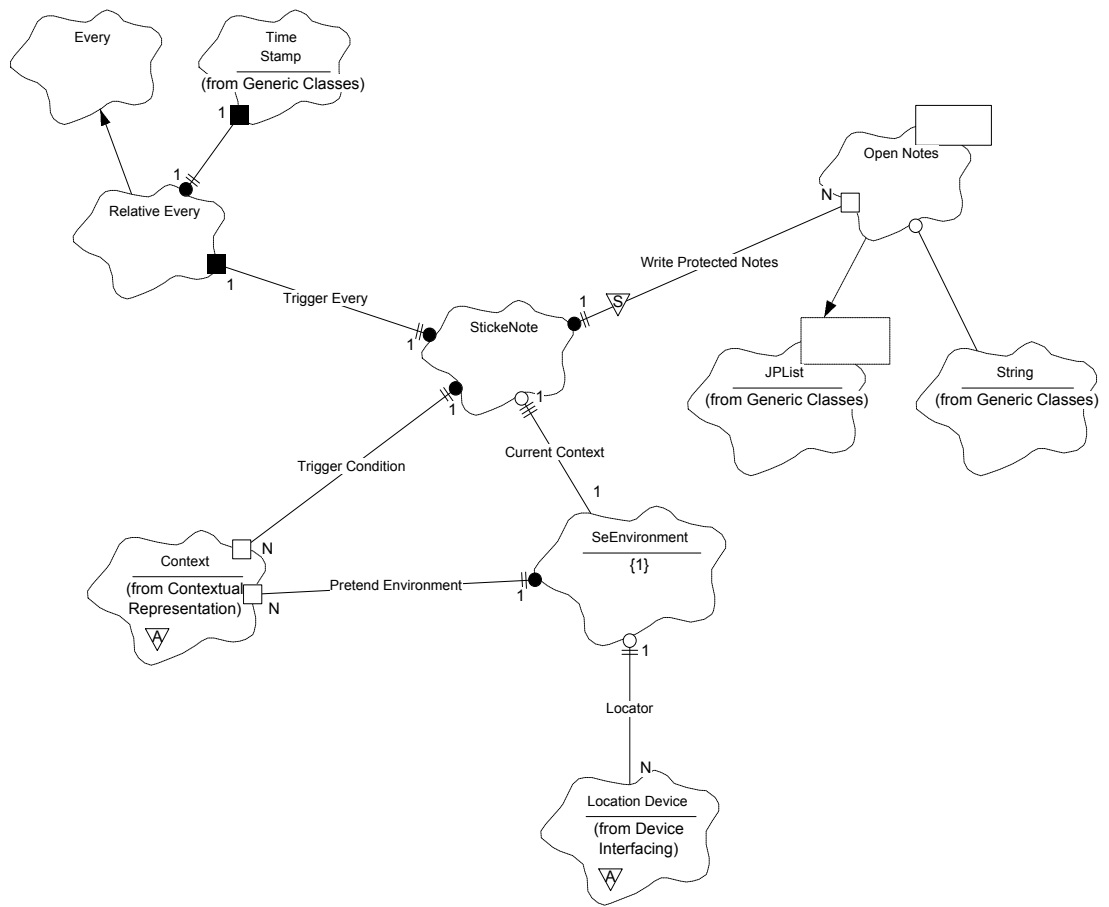
Figure 57. Class diagram for the stick-e note management category.
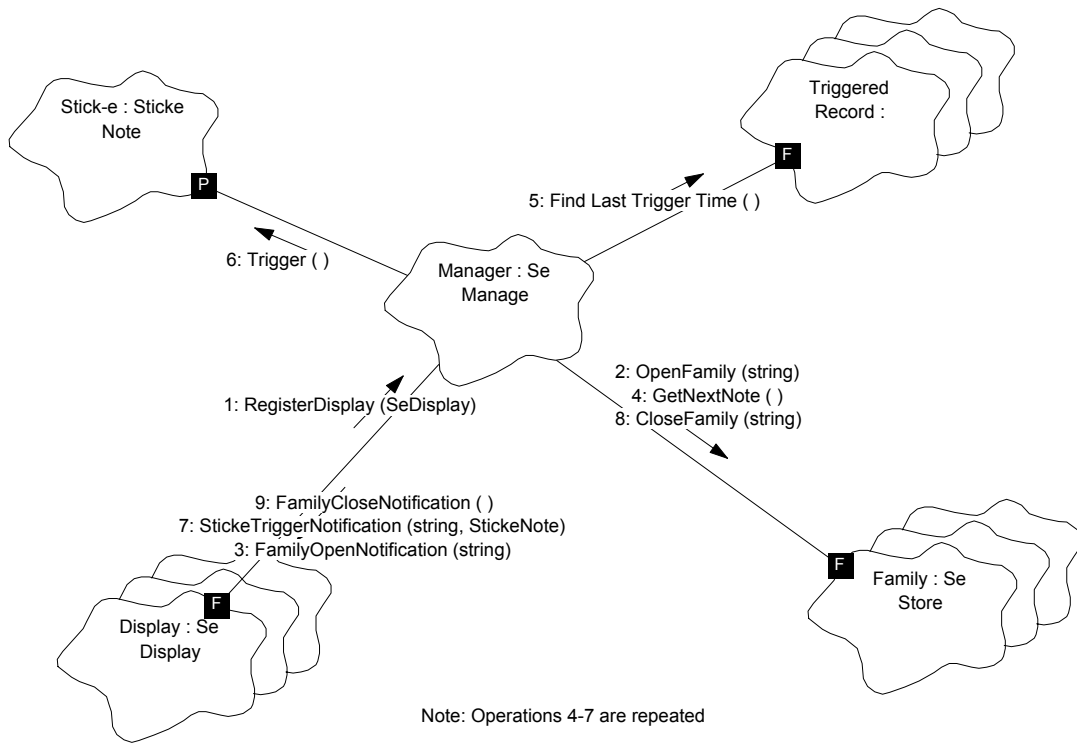
Figure 58. Object interaction diagram for the family management classes.
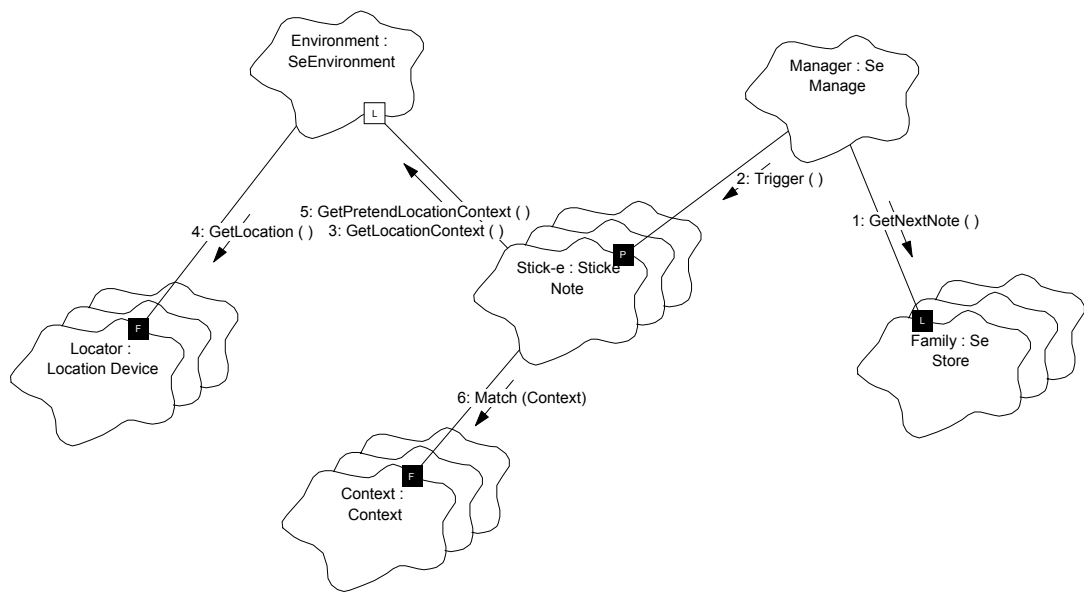
Figure 59. Object interaction diagram for the stick-e management classes.

Appendix B:

Sticke-e Note Desktop

Authoring Tool

This appendix illustrates through a series of screenshots some of the features of the PC-based stick-e note management and simulation tool. Using this application, simulations of different environments were performed with test stick-e notes and families. This enabled the trigger checking and model-view-controller mechanisms to be tested in addition to the storage, retrieval and manipulation of stick-e notes with their attached content and contexts. Additionally a GPS receiver (in simulator-mode) was attached to the computer to facilitate the testing of 'real' location data in addition to pretend contexts.
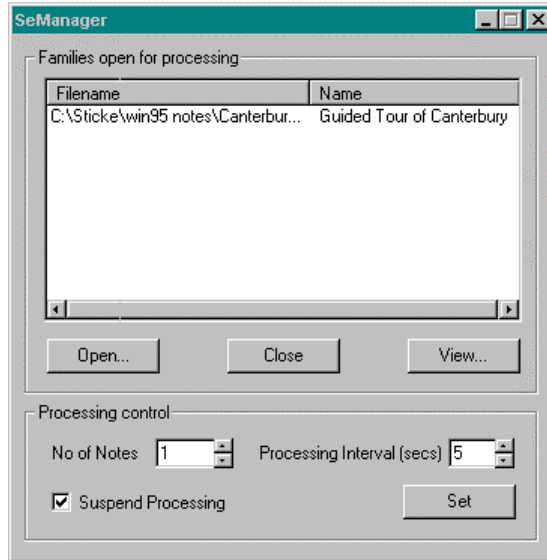
Figure 60. Direct control over the SeManage processing parameters (such as frequency and intensity of processing) and a view into the internal status of SeManage.
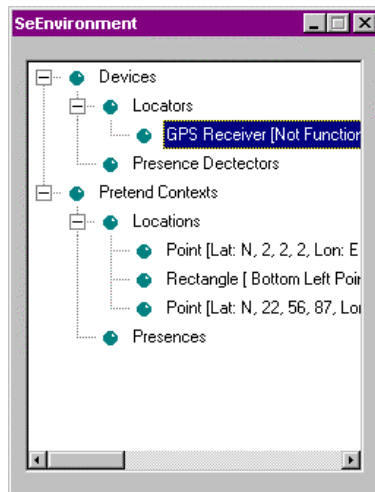
Figure 61. Environmental status panel through which the devices and pretend contexts that are present within the SeEnvironment object can be viewed, created, updated, amended and deleted.
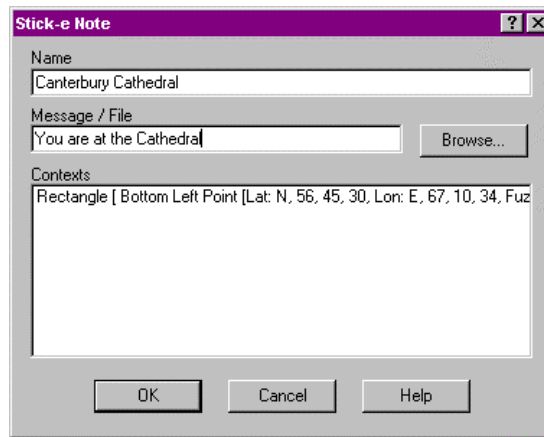
Figure 62. Facility for creating and editing stick-e notes, entering or referencing content for the note, and attaching the stick-e note to a context(s).
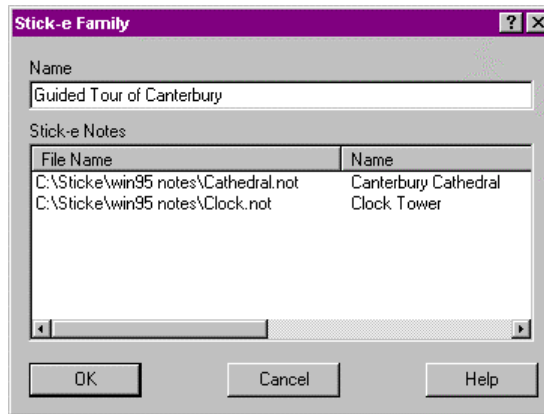
Figure 63. Creation and maintenance of families (into which stick-e notes can be later added or removed).
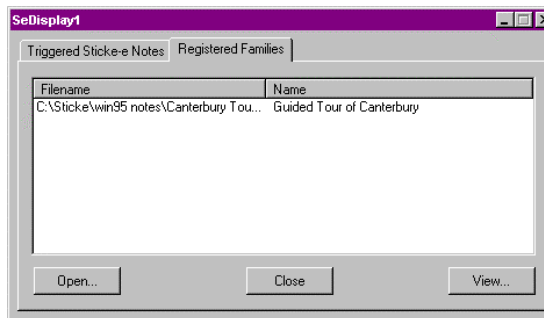
Figure 64. An implementation of a SeDisplay class is provided, which lets the user directly register families for processing, or respond to open-family notification messages from the SeManage. Any incoming triggered stick-e notes are made available for the user to view. Many SeDisplays can be open at once to test the *one to many* relationship from controller to view(s).

Appendix B: Stick-e Note Desktop Authoring Tool

Appendix C:
Usability Survey Questionnaire

Earthwatch "Computing in the Field" Questionnaire

You have been one of the first people to carry out the various fieldwork activities using a PalmPilot handheld computer. This is a new method of data collection that we are pioneering, so we are most interested to receive feedback from you on your experiences with this new technology.

For us the PalmPilots offer many benefits, such as ensuring the consistency of data collected and providing a quick and transcription-error-free method of entry into a spreadsheet on the day of collection. The latter ability enables us to start processing the data immediately and to give you timely feedback on any preliminary results before you leave. Downloading onto the laptop computer also enables us to implement a backup strategy to ensure the safety of the data; the worst-case scenario of equipment failure would only result in a single day's data being lost.

In this questionnaire we are interested in the viewpoint of you, the fieldwork volunteer, and your experience of using the PalmPilot in the field. From the information collected in these questionnaires we hope to discover how the PalmPilots have impacted on your fieldwork experience and to identify areas where we can make improvements. We are also interested in correlating people's views with any background or previous experience in ecology/environmental work or computing.

If you are interested in learning about the results of this questionnaire please fill in your email address in the personal details section, below.

Your Personal Details

Name:

Email Address:

Sex (M/F):

Age (20s 30s 40s 50s 60s 70s):

Occupation:

Nationality:

Your Fieldwork Background

1. Have you been involved in any other Earthwatch projects? If so, which ones?

2. Do you have any ecology/environmental *work experience* or *qualifications*? If so, please describe them.

3. What previous ecology/environmental *fieldwork experience* do you have?

4. What other general fieldwork experience do you have? For example, archaeological fieldwork.

Your Computing Background

5. Do you have any computing work experience or qualifications? If so, please describe them.

6. Do have a computer at home? If so, what are the main things you use it for?

7. Have you used a handheld computer before? If so, what make/model (e.g. PalmPilot, Psion Series 5, Casio Windows CE, etc.) and for how long and for what purpose?

Your Experience using the PalmPilot

8. How did you feel about using the PalmPilot before you arrived?

   ❑ Looking forward to using it a lot
   ❑ Looking forward to using it a little
   ❑ Not worried one way of the other
   ❑ A little worried about using it
   ❑ Very worried about using it

9. Was there enough training to use the PalmPilots?

   ❑ Too much
   ❑ A bit too much
   ❑ Just the right amount
   ❑ Too little
   ❑ Much too little

10. If you went on a similar project in the future how would you feel about using a PalmPilot again?

  ❑ Looking forward to using it a lot

  ❑ Looking forward to using it a little

  ❑ Not worried one way of the other

  ❑ A little worried about using it

  ❑ Very worried about using it

11. How many sessions of using a PalmPilot did it take before you felt comfortable in using it? By 'session' we mean using the PalmPilot to carry out a particular day's activities. Please indicate if you were comfortable using the PalmPilot before the first session (i.e. immediately after the training) or are still not comfortable in using it.

12. Given the choice, would you now prefer to use paper, pen, map, and stopwatch, or PalmPilot and GPS to collect your data in the field? Please explain your choice.

13. Did you or your partner do most of the PalmPilot work, or was it roughly equal? If not equal was there any particular reason why?

14. What things, if any, did you particularly like about using the PalmPilot?

15. What things, if any, did you particularly dislike about using the PalmPilot?

16. How easy did you find using Graffiti? (Graffiti is handwriting-like method of entering letters and numbers into the PalmPilot). Do not tick any boxes if you did not use Graffiti.

  ❑ Very easy

  ❑ Easy

  ❑ Okay

  ❑ Difficult

  ❑ Very difficult

17. How easy was it to use on-screen keyboard? Do not tick any boxes if you did not use the keyboard.

  ❑ Very easy

  ❑ Easy

  ❑ Okay

  ❑ Difficult

  ❑ Very difficult

18. How easy was it to choose the correct template(s) for your activity?

  ❑ Very easy

  ❑ Easy

  ❑ Okay

  ❑ Difficult

  ❑ Very difficult

19. How easy was it to enter all the data?

  ❑ Very easy

  ❑ Easy

  ❑ Okay

  ❑ Difficult

  ❑ Very difficult

20. Were there any types of data (e.g. numbers, picklists, GPS location, line of text, etc.) that you found particularly easy to enter?

21. Were there any types of data (e.g. numbers, picklists, GPS location, line of text, etc.) that you found particularly hard to enter?

22. Did you find any terminology confusing? If so what?

23. Were there any difficulties you had using the PalmPilot that you haven't already mentioned?

24. Do you have any suggestions for improvements that could be made?

25. Generally speaking, how did you find using the PalmPilot on the whole?

- ❑ Very easy
- ❑ Easy
- ❑ Okay
- ❑ Difficult
- ❑ Very difficult

26. How many resets do you estimate you had to perform?

27. Did you have any trouble linking the PalmPilot to the GPS receiver? If so, please describe.

28. Please write any other comments you would like to make in the space below.

Thank you very much for taking time to complete this questionnaire.

Appendix D:
Multiple-Choice Codified
Response Sheet

[The columns of the sheet are spread out over the following pages.

The "Volunteer No" column is repeated for readability]

Team No.	Volunteer No	Name	Email	Sex	Age
s	s1	Sally Burnett	si_burnett@hotmail.com	f	20
s	s2	Kathy Pinkney	kathy@mepinkney.freeserve.co.uk	f	20
1	1	Kathy White	kwhite@earthwatch.org	f	30
1	2	Anne Levin	anne_levin@prodigy.net	f	20
1	3	Veronica Anderson	veronica_anderson@yahoo.com	f	20
1	4	William Lemp	complaw@epconline.com	m	40
1	5	Steve Readinger	-	m	50
1	6	Brian Robbins	brobbins@loeb.com	m	30
1	7	Trish Levin	tlevin@raystoneinc.com	f	50
1	8	Dianne Clapp	davesong@aol.com	f	50
1	9	Samanha Gerry	caninenurse@juno.com	f	20
1	10	Annette Purdue	-	f	20
1	11	Carola hallay	carola.hallay@unilever.com	f	30
2	12	Ronda Planck-Preston	czrhinos@aol.com	f	30
2	13	Burke Johnson	bjohnson@carteransley.com	m	30
2	14	Cara Cupit	cara7471@aol.com	f	20
2	15	Monte Shouse	-	m	50
2	16	Diane Grauel	patrusrach@aol.com	f	20
2	17	Duane Nickell	dnic310@aol.com	m	40
2	18	Sarah Howell	sah39@hermes.cam.ac.uk	f	20
2	19	Carola Braune	carola.heiner@t-online.de	f	50
2	20	Heiner Stickann	carola.heiner@t-online.de	m	40
3	21	Sabine McCarthy	sabine_peter_99@hotmail.com	f	30
3	22	Peter McCarthy	sabine_peter_99@hotmail.com	m	20
3	23	John McCracken	pachomama@earthlink.net	m	20
3	24	Carrie McCracken	pachomama@earthlink.net	f	20
3	25	Bonnie Bahn	bonnie.bahn@hmsc.orst.edu	f	50
3	26	Kathryn Dunn	kdunn@syl.sj.nec.com	f	20
3	27	Yuka Maezawa	yuka@cim.ed.fujitsu.co.jp	f	30
3	28	Stuart Grayson	stuart.grayson@bluewin.ch	m	40
3	29	"T" St. John	tstjohn@alliantfs.com	f	30
4	30	Sharon Castner	SharenC@home.com	f	50
4	31	Gary Kaufman	GarySharon@home.com	m	50
4	32	Nancy Ronan	NLMnyc@aol.com	f	40
4	33	Felix Patton	bioscope@zetnet.co.uk	m	40
4	34	Kathy Smith	Kathy_Smith@hotmail.com	f	20
4	35	Ginny Hounsell	ginny5@earthlink.net	f	40
4	36	Sue Davis	sue.davis@talk21.com	f	30
4	37	Pat Williamson	PATWMSN@worldnet.att.net	f	50
4	38	Bruni Tessmer	btessmei@ford.com	f	50

Vol. No	Occupation	Nationality	Fieldwork Experience (1-4)	Computing Experience (1-4)
s1	Student	British	3	2
s2	Student	British	4	3
1	Dept Director, Earthwatch Centre for Field Research	USA	4	3
2	Admin Assistant	USA	2	3
3	Japanese Marketing Liason	USA	2	4
4	Attorney	USA	3	1
5	Family Physician	USA	1	1
6	Financial Officier of Law Firm	USA	2	3
7	CPA/Finance Director	USA	2	2
8	Nurse	USA	1	2
9	Veterinary Nurse	USA	1	3
10	Veterinary Nurse	USA	1	2
11	Secretary	German	1	2
12	Cincinnati Zoo Education Department	USA	2	3
13	Attorney	USA	2	2
14	Teacher	USA	1	2
15	Retired	USA	2	1
16	Retail Petshop Manager	USA	1	2
17	Teacher	USA	2	2
18	Student	British	2	2
19	Teacher	German	2	3
20	Teacher	German	2	3
21	Pharmacist	Swiss	1	2
22	Engineer	British	2	1
23	Wildlife Rehabilitator	USA	1	2
24	Student	USA	3	3
25	Secretary	USA	2	3
26	Marketing Assistant	USA	2	3
27	Programmer	Japanese	2	4
28	IT Consultant	British	2	4
29	Marketing	USA	1	3
30	Flight Attendant	USA	1	2
31	Airport Management	USA	2	2
32	Theatre	USA	1	2
33	Development Manager, Agribusiness	British	2	3
34	Librarian	Australian	2	2
35	Theatre	USA	1	2
36	Manager (BT)	British	2	3
37	Court Reporter	USA	2	2
38	P.A.	German	1	2

Appendix D: Multiple Choice Codified Response Sheet

Vol. No	6. Handheld Computer User (y/n)	7. Handheld Computer Model	8. Feel before (1-5)	9. Enough Training (1-5)
s1	n	-	-	3
s2	y	PalmPilot	1	3
1	n	-	1	3
2	n	-	3	3
3	y	PalmPilot	1	3
4	n	-	3	3
5	n	-	3	4
6	n	-	3	3
7	n	-	1	3
8	n	-	4	4
9	n	-	3	3
10	n	-	4	3
11	n	-	2	4
12	y	PalmPilot	3	4
13	n	-	3	4
14	n	-	1	3
15	n	-	4	3
16	n	-	3	3
17	n	-	3	3
18	y	IBM Workpad	2	4
19	n	-	1	3
20	n	-	1	3
21	n	-	3	3
22	n	-	3	3
23	n	-	3	3
24	n	-	3	3
25	n	-	2	3
26	n	-	2	3
27	n	-	4	2
28	n	-	3	3
29	n	-	1	3
30	n	-	2	4
31	n	-	3	3
32	n	-	1	3
33	n	-	3	3
34	n	-	3	3
35	n	-	4	4
36	y	Psion 3c, 5, 5mx	1	3
37	n	-	4	4
38	n	-	3	4

Vol. No	10. Feel after (1-5)	Feeling Improvement	11. Comfortable after (1-5)	12. Preferred System (pp, e, man)
s1	1	-	1	pp
s2	1	0	1	pp
1	1	0	1	pp
2	1	2	1	pp
3	1	0	1	pp
4	2	1	3	pp
5	1	2	3	pp
6	2	1	2	pp
7	1	0	1	pp
8	3	1	-	pp
9	1	2	1	pp
10	1	3	1	pp
11	1	1	1	pp
12	1	2	1	pp
13	3	0	3	pp
14	1	0	1	pp
15	3	1	5	pp
16	3	0	3	pp
17	2	1	3	pp
18	1	1	1	pp
19	1	0	1	pp
20	1	0	1	pp
21	1	2	1	pp
22	2	1	1	pp
23	1	2	1	pp
24	1	2	1	pp
25	2	0	2	pp
26	2	0	1	pp
27	1	3	2	pp
28	3	0	2	pp
29	2	-1	3	pp
30	1	1	2	pp
31	1	2	1	pp
32	1	0	1	pp
33	3	0	1	pp
34	1	2	1	pp
35	3	1	2	pp
36	1	0	1	pp
37	3	1	4	pp
38	3	0	3	pp

Vol. No	13. Work division (y, e, p)	16. Graffiti ease of use (1-5)	17. Keyboard ease of use (1-5)
s1	y	1	2
s2	y	1	1
1	e	1	1
2	e	2	2
3	e	1	1
4	e	2	2
5	e	2	-
6	e	3	1
7	y	3	1
8	p	-	-
9	e	-	1
10	e	-	2
11	e	2	1
12	y	2	1
13	p	4	2
14	e	-	2
15	p	3	3
16	e	3	1
17	y	2	2
18	e	2	2
19	y	-	1
20	p	-	1
21	e	2	-
22	e	2	-
23	e	1	1
24	p	3	2
25	e	-	4
26	e	2	1
27	e	2	1
28	e	4	2
29	e	2	1
30	y	2	1
31	p		
32	y	3	2
33	p		
34	e	3	2
35	e	3	1
36	y	3	2
37	p	3	2
38	e	4	1

Appendix D: Multiple Choice Codified Response Sheet

Vol. No	18. Template selection ease of use (1-5)	19. All data entry ease of use (1-5)	25. Using PP ease of use (1-5)
s1	1	2	1
s2	1	1	1
1	1	2	1
2	1	1	1
3	1	2	1
4	1	2	2
5	2	1	2
6	1	3	2
7	1	1	1
8	2	2	3
9	1	1	1
10	1	2	1
11	2	2	2
12	1	2	2
13	3	3	2
14	1	1	2
15	3	2	2
16	2	2	2
17	1	2	1
18	1	2	2
19	1	1	1
20	1	1	1
21	1	1	1
22	1	1	1
23	1	2	2
24	2	1	2
25	2	2	3
26	1	1	1
27	2	2	2
28	2	2	2
29	1	1	1
30	1	2	1
31			
32	2	1	1
33			
34	2	3	2
35	1	3	1
36	1	2	1
37	2	2	2
38	2	2	3

Vol. No	26. No of resets	27. GPS Trouble (y/n)
s1	1	n
s2		n
1	20	y
2		n
3	20	n
4	48	n
5	6	y
6	24	n
7	28	n
8	10	n
9	24	n
10	24	n
11	24	y
12	10	n
13	8	n
14	20	n
15	8	n
16	15	n
17	24	n
18	12	n
19	36	n
20	36	n
21	7	n
22	7	n
23	8	n
24	15	n
25	6	y
26	5	n
27	1	n
28	3	n
29	1	n
30	20	n
31		
32	4	y
33		n
34	2	n
35	6	n
36	50	n
37	0	n
38	0	n

Appendix E:

Codification Scheme

For Numerically Coded Questions:

	1	2
Fieldwork Experience (1-4)	none (or nearly none)	1-2 previous projects
Computing Experience (1-4)	none (or nearly none)	occassional use (word, email, etc.)
8. Feel before (1-5)	looking forward to using it a lot	looking forward to use it a little
9. Enough Training (1-5)	too much	a bit too much
10. Feel after (1-5)	looking forward to using it a lot	looking forward to use it a little
11. Comfortable after (1-5)	first time	first day
16. Graffiti ease of use (1-5)	very easy	easy
17. Keyboard ease of use (1-5)	very easy	easy
18. Template selection ease of use (1-5)	very easy	easy
19. All data entry ease of use (1-5)	very easy	easy
25. Using PP ease of use (1-5)	very easy	easy

3	4	5
many previous projects	Full time (professional)	-
profficient user	computer expert	-
not worried one way or the other	a little worried about using it	very worried about using it
just the right amount	too little	much too little
not worried one way or the other	a little worried about using it	very worried about using it
2-3 days	one week	never
okay	difficult	very difficult
okay	difficult	very difficult
okay	difficult	very difficult
okay	difficult	very difficult
okay	difficult	very difficult

For Textually Coded Questions:

6. Handheld Computer User (y/n)	y = yes, n = no
7. Handheld Computer Model	name of make and model
12. Preferred System (pp, e, man)	pp = PalmPilot system, e = either, man = manual system
13. Work division (y, e, p)	y = you did most, e = both equal, p = partner did most
27. GPS Trouble (y/n)	y = yes, n = no

Appendix F:

Categorised Response Sheet

[This sheet is spread out over the following pages in a row-first manner.

Row and column headings are repeated for readability.]

	Total	s1	s2	38	37	36	35	34	33	32	31
12a. Prefer PalmPilot system because											
less chance for error	5						1				
data entry faster	9		1								1
transcription quicker & easier	6		1			1					
prompting help	2										
more comfortable	4							1			
more efficient / less work & writing	8				1						
consistency in collecting data	2		1								
work is easier / simpler	4									1	
weather-proof	1										
12b. Prefer manual system because											
13. Partner inequality reasons											
differing ability	4				1						
preference	7				1	1				1	
14. PalmPilot likes											
data prompts in template	6					1					
input data standardisation	4					1			1		
easy to enter data in forms	9							1			
easy to keep neat (e.g. correct errors)	4								1		
quick to enter data	7						1				
compact device - easy to carry	12				1		1		1		
less mistakes	2										
easy to download	6							1	1		
more efficient / less work & writing	5				1						
GPS connection	2										
helped non-English speaker through Picklists etc.	1										
can walk and use it at the same time	1						1				
everything	1										
15. PalmPilot dislikes											
Resets	16	1	1			1		1			
losing data	1										
speed of entry	6					1		1		1	
general data entry tedium	1										
hard to read screen at times	5								1		
easy to lose pen	1										
linking up GPS took time	1										
non-standard comments time-consuming/difficult	1										
worried about batteries running down	2										

	Total	s1	s2	38	37	36	35	34	33	32	31
20. Easy data types to enter											
all	10		1							1	
GPS location	17					1	1	1			
picklists	11				1	1		1			
numbers	4							1			
bearing	1										
dung count	1										
21. Hard data types to enter											
comments	1										
GPS (because of cable)	1										
numbers	2				1						
picklists	1										
text	5					1					
time values that were not the current time	1										
22. Confusing terminology											
23. PalmPilot difficulties											
batteries run down	2										
keeping up with behaviour data entry	1										
slow at times	2									1	
jumped to one-handed mode	1										
downloading difficult	1										
wanted to compare Pilot & GPS locations	1										
24. Suggestions for improvements											
better picklists (e.g. multi-choice)	4		1								
default data values	1										
less cumbersome cables	2										
graphical forms (e.g. footprints)	1										
speeded up	2							1			
less steps to filling in form (1 button not 3)	1	1									
wireless connection between Palm & GPS	1										
field name displayed in edit dialogs	1										
automatic sequential numbering where appropriate	1										
download straight into Excel	1		1								
27. GPS trouble											
GPS pin broke	1										
Cable connection loose	4									1	
Created note before GPS was on	1										

Appendix F: Categorised Response Sheet

12a. Prefer PalmPilot system because

	30	29	28	27	26	25	24	23	22	21
less chance for error		1								
data entry faster							1			
transcription quicker & easier		1								
prompting help										
more comfortable										
more efficient / less work & writing			1							
consistency in collecting data										
work is easier / simpler			1			1				
weather-proof	1									

12b. Prefer manual system because

13. Partner inequality reasons

	30	29	28	27	26	25	24	23	22	21
differing ability										
preference	1	1				1				

14. PalmPilot likes

	30	29	28	27	26	25	24	23	22	21
data prompts in template		1								
input data standardisation										
easy to enter data in forms		1								
easy to keep neat (e.g. correct errors)								1		
quick to enter data										
compact device - easy to carry	1		1					1		
less mistakes						1				
easy to download							1			
more efficient / less work & writing	1							1		
GPS connection					1					
helped non-English speaker through Picklists etc.				1						
can walk and use it at the same time										
everything										

15. PalmPilot dislikes

	30	29	28	27	26	25	24	23	22	21
resets	1					1				
losing data										
speed of entry							1			
general data entry tedium										
hard to read screen at times		1								
easy to lose pen										
linking up GPS took time										
non-standard comments time-consuming/difficult										
worried about batteries running down		1								1

	30	29	28	27	26	25	24	23	22	21
20. Easy data types to enter										
all		1						1	1	
GPS location	1		1		1		1			1
picklists	1				1		1			
numbers	1				1					
bearing										
dung count										
21. Hard data types to enter										
comments										
GPS (because of cable)										
numbers										
picklists										1
text	1				1	1	1			
time values that were not the current time			1							
22. Confusing terminology										
23. PalmPilot difficulties										
batteries run down							1			
keeping up with behaviour data entry										
slow at times										
jumped to one-handed mode										
downloading difficult						1				
wanted to compare Pilot & GPS locations			1							
24. Suggestions for improvements										
better picklists (e.g. multi-choice)									1	
default data values										
less cumbersome cables										
graphical forms (e.g. footprints)										
speeded up										
less steps to filling in form (1 button not 3)										
wireless connection between Palm & GPS					1					
field name displayed in edit dialogs	1									
automatic sequential numbering where appropriate	1									
download straight into Excel										
27. GPS trouble										
GPS pin broke										
Cable connection loose										
Created note before GPS was on						1				

	20	19	18	17	16	15	14	13	12	11
12a. Prefer PalmPilot system because										
less chance for error										
data entry faster			1		1		1			
transcription quicker & easier										
prompting help										
more comfortable	1	1							1	
more efficient / less work & writing		1	1		1	1				
consistency in collecting data										
work is easier / simpler	1									
weather-proof										
12b. Prefer manual system because										
13. Partner inequality reasons										
differing ability						1				
preference									1	
14. PalmPilot likes										
data prompts in template				1						
input data standardisation										
easy to enter data in forms			1		1				1	
easy to keep neat (e.g. correct errors)										
quick to enter data						1	1			
compact device - easy to carry					1					
less mistakes										
easy to download							1			
more efficient / less work & writing					1					
GPS connection										
helped non-English speaker through Picklists etc.										
can walk and use it at the same time										
everything										
15. PalmPilot dislikes										
resets				1		1		1	1	
losing data								1		
speed of entry				1						
general data entry tedium										
hard to read screen at times					1					
easy to lose pen						1				
linking up GPS took time				1						
non-standard comments time-consuming/difficult										
worried about batteries running down										

	20	19	18	17	16	15	14	13	12	11
20. Easy data types to enter										
all		1					1			
GPS location					1			1	1	
picklists					1			1		1
numbers										
bearing										
dung count										
21. Hard data types to enter										
comments								1		
GPS (because of cable)				1						
numbers										
picklists										
text										
time values that were not the current time										
22. Confusing terminology										
23. PalmPilot difficulties										
batteries run down							1			
keeping up with behaviour data entry									1	
slow at times						1				
jumped to one-handed mode				1						
downloading difficult										
wanted to compare Pilot & GPS locations										
24. Suggestions for improvements										
better picklists (e.g. multi-choice)										
default data values										
less cumbersome cables								1		
graphical forms (e.g. footprints)										
speeded up									1	
less steps to filling in form (1 button not 3)										
wireless connection between Palm & GPS										
field name displayed in edit dialogs										
automatic sequential numbering where appropriate										
download straight into Excel										
27. GPS trouble										
GPS pin broke										
Cable connection loose					1				1	
Created note before GPS was on										

	10	9	8	7	6	5	4	3	2	1
12a. Prefer PalmPilot system because										
less chance for error		1		1				1		
data entry faster						1		1	1	
transcription quicker & easier				1		1		1		
prompting help		1						1		
more comfortable										
more efficient / less work & writing				1		1				
consistency in collecting data					1					
work is easier / simpler										
weather-proof										
12b. Prefer manual system because										
13. Partner inequality reasons										
differing ability			1	1						
preference										
14. PalmPilot likes										
data prompts in template			1						1	1
input data standardisation					1					1
easy to enter data in forms	1						1	1	1	
easy to keep neat (e.g. correct errors)						1				1
quick to enter data	1					1			1	1
compact device - easy to carry	1			1	1		1			1
less mistakes								1		
easy to download					1		1			
more efficient / less work & writing				1						
GPS connection				1						
helped non-English speaker through Picklists etc.										
can walk and use it at the same time										
everything			1							
15. PalmPilot dislikes										
resets	1	1			1		1	1	1	
losing data										
speed of entry					1					
general data entry tedium				1						
hard to read screen at times	1		1							
easy to lose pen										
linking up GPS took time										
non-standard comments time-consuming/difficult									1	
worried about batteries running down										

20. Easy data types to enter

	10	9	8	7	6	5	4	3	2	1
all		1		1				1		
GPS location		1			1	1	1	1		1
picklists			1					1		
numbers							1			
bearing						1				
dung count										1

21. Hard data types to enter

	10	9	8	7	6	5	4	3	2	1
comments										
GPS (because of cable)										
numbers			1							
picklists										
text										
time values that were not the current time										

22. Confusing terminology

23. PalmPilot difficulties

batteries run down

keeping up with behaviour data entry

slow at times

jumped to one-handed mode

downloading difficult

wanted to compare Pilot & GPS locations

24. Suggestions for improvements

	10	9	8	7	6	5	4	3	2	1
better picklists (e.g. multi-choice)				1		1				
default data values		1								
less cumbersome cables		1								
graphical forms (e.g. footprints)		1								
speeded up										
less steps to filling in form (1 button not 3)										
wireless connection between Palm & GPS										
field name displayed in edit dialogs										
automatic sequential numbering where appropriate										
download straight into Excel										

27. GPS trouble

	10	9	8	7	6	5	4	3	2	1
GPS pin broke									1	
Cable connection loose						1				
Created note before GPS was on										

Appendix G:

Narrative Responses

S1. Kathy Pinkney

Notes are time-stamped and its easy to collect time-budget data.

1. Kathy White

GPS is great but probably doesn't help give you a good physical understanding of where you are unlike a manual map and compass.

Need a more efficient method of recording giraffe observations. New template for each tree is cumbersome.

Good for data standardisation, parameterisation, and prompting.

Data entry speed is faster good (faster than manual system).

I think PalmPilots are a great way to have volunteers collect data.

It will be interesting to see the robustness over the course of a summer in comparison to other equipment. Also interesting to see if volunteers adapt to using the technology or if anyone is overwhelmed.

2. Anne Levin

Stickepad should auto-scroll to last note recorded.

3. Veronica Anderson

Would be nice to have default values for some fields.

4. William Lemp

Would be nice to have multiple options selectable from a picklist.

It is a great tool for field research.

6. Brian Robbins

Concept seems sound and exporting of data directly to spreadsheet is great!

Tech support very friendly & helpful.

7. Trish Levin

Suggest graphical multi-menu-picklist idea for faster entry of behaviours.

I'm really excited about your work. Good luck!

9. Samantha Gerry

I enjoyed using both pieces of equipment and found them easy to use, it was an effort to think of suggestions.

10. Annette Purdue

It was a good learning experience for me to use the GPS and PalmPilot. I hope to use them in the future. Thanks for your patience!

12. Ronda Planck-Preston

Palm/GPS is the best way to get the best data for research.

13. Burke Johnson

Overall, it was easy and a good experience.

Had some concern about switching from template to template when it asks about deleting data.

For you the best thing is probably that you don't have to read my handwriting!

16. Diane Grauel

Had fun and enjoyed the chance to use the PalmPilot and GPS.

17. Duane Nickell

Good luck, Jason! It's been great knowing you!

25. Bonnie Bahn

Keeping batteries charged and spares available isn't done daily and it needs to be.

27 Stuart Grayson

Organisation improvement: each team should have a PalmPilot allocated and the team is responsible for ensuring that the batteries are OK and that they have back-ups.

Dislike: Trying to capture data originally captured on paper, i.e. having to override the time entry which is picked up from the clock.

Sometimes we wanted to use a GPS location already in the PalmPilot and compare with current GPS – but the PalmPilot stores the co-ordinates in non-UTM format.

28. "T" St.John

'h' hard to write at times.

30. Sharon Castner

Jason, thanks I really enjoyed using this. You might contact Dave Balfour at Hlu Hluwe/omfolozi in South Africa. All his data was done with clipboards and pencils & paper and compass headings. He may be very receptive to this.

Sometimes keys pressed on the on-screen keyboard did not respond [*almost certainly was due to screen mis-calibration*].

In 1998 I used paper/pencil and clipboard in the drizzle walking down/up mountains. Rain washed some data away and another girl sweated her data off.

33. Felix Patton

Good System but there must be some way of preventing complete loss of data.

34. Kathy Smith

Took too much time when doing rhino & giraffe behaviour – they move too fast and the PP moves too slow.

35. Ginny Hounsell

If a couple of the templates were tightened up it would make the recording even faster and easier.

I'm glad I got to use the PalmPilot since I'm not experienced in computer work. I learned a little more and overcame some reservations.

36. Sue Davis

It couldn't go fast enough, i.e. I could tap quicker than the screen refreshed. Also after crashes I had to page through all notes to get to the last one I was working on.

Psion is better for data entry but not when standing up. Comments took a long time to enter.

Ground cover – Kathy said one thing, Alan another.

No note for Morani's flatulence.

Most fatal errors were on vegetation trees which is also the longest spreadsheet.

I would like to try the system with other Earthwatch projects. I would prefer to get the fatal errors sorted first – I call tell when it is going to happen as previous boxes do not disappear when they should so usually I get time to save before it goes. But it is the same error under the same conditions usually (high speed entry or high number of notes).

37. Pat Williamson

Didn't like walking and typing. Pencil should have been attached.

Before a team goes out it would be perhaps more efficient if all the parties were comfortable with the equipment. Since I don't really enjoy using computers I left it up

to someone else and when she was not there my other partner and I struggled and felt very incompetent. I will be sure that doesn't happened to me again. I felt like I wasted Kathy's time and some of our own. I should not have been so lazy in relying on someone else.

Appendix H:

Equipment Fault Log

Summary:

Total Resets	88
Total Notes	4162
Notes per reset	47.3
Paper used	9
Data Lost	2
PP Locked Up	9
GPS Trouble	1

[The sheet is spread out over the next few pages in a row-first manner.

Row and column headings are repeated for readability.]

Date	Pair No	Activity	PalmPilot No	GPS No	Cable No	No of Resets	GPS Link-up Trouble	PP Locked Up
28-Jul-99	1	a	5			0	n	n
28-Jul-99	2	f	3			3	n	n
28-Jul-99	3	d	4			0	n	n
28-Jul-99	4	c1	2		3	0	y	n
29-Jul-99	1	b	1			0	n	n
29-Jul-99	2	a	3		4	2	n	n
29-Jul-99	3	f	2			0	n	n
29-Jul-99	4	d	4	3	3	1	n	n
29-Jul-99	5	d	5	3	3	1	n	n
30-Jul-99	1	c1	2		2	0	n	n
30-Jul-99	2	b	3		4	0	n	n
30-Jul-99	3	a	1		3	0	n	y
30-Jul-99	4	f	4			1	n	n
31-Jul-99	2	c1	2		3	0	n	n
31-Jul-99	3	b	3		4	0	n	n
31-Jul-99	4	a	4	4	2	3	n	n
1-Aug-99	1	d	1	1	1	2	n	n
1-Aug-99	3	c1	2	3	2	0	n	n
1-Aug-99	4	b	4	4		0	n	n
2-Aug-99	1	b	1	1	1	1	n	n
2-Aug-99	2	d	3		2	5	n	n
2-Aug-99	4	c2	4	3		1	n	n
3-Aug-99	1	f	2			3	n	n
3-Aug-99	2	e	3			10	n	n
3-Aug-99	3	d	1	4		2	n	n
3-Aug-99	4	d	4	2		3	n	n
4-Aug-99	1	a	1			2	n	n
4-Aug-99	2	f	3			5	n	n
4-Aug-99	3	e	2		2	6	n	y
5-Aug-99	1	b	1		1			
5-Aug-99	3	f	3			5		y
5-Aug-99	4	e	2	2		3	n	n
6-Aug-99	1	c2	1	1	1	0	n	n
6-Aug-99	3	a	4			3	n	n
6-Aug-99	4	f	2			2	n	n
7-Aug-99	3	b	2	2		0	n	n
7-Aug-99	4	a	4			2	n	n
25-Aug-99	1	a	2	3	1	4	n	y
25-Aug-99	2	f	3			0	n	n
25-Aug-99	3	d	1	2	2	2	n	y
25-Aug-99	4	c1	5		3	0	n	n
26-Aug-99	1	b	4	4	3	0	n	n
26-Aug-99	2	a	3		1	2	n	n
26-Aug-99	4	d	5	3	2	0	n	n

Appendix H: Equipment Fault Log

Date	Pair No	Activity	PalmPilot No	GPS No	Cable No	No of Resets	GPS Link-up Trouble	PP Locked Up
27-Aug-99	2	c1	2	2	1	0	n	n
27-Aug-99	3	a	4		3	1	n	y
27-Aug-99	4	f	5			0	n	n
28-Aug-99	2	d	4		3	3	n	n
28-Aug-99	3	b	2	2	1	0	n	n
28-Aug-99	4	a	5	4	2	0	n	n
29-Aug-99	4	b	5	4	3	0	n	n
30-Aug-99	1	d	3		2	2	n	n
30-Aug-99	2	b	4	3	3	0	n	n
30-Aug-99	3	c1	2	2	1	0	n	n
31-Aug-99	1	f	3		0	0	n	n
31-Aug-99	2	e	5	4	3	1	n	n
31-Aug-99	3	g	2		1	0	n	n
31-Aug-99	4	b	1	3	4	0	n	n
1-Sep-99	1	a	2	3	2	0	n	
1-Sep-99	2	f	3			0	n	n
1-Sep-99	3	e	1	4	4	1	n	y
1-Sep-99	4	g	5			0	n	n
2-Sep-99	1	e	3	2		2	n	n
2-Sep-99	2	a	2		3	1	n	n
2-Sep-99	3	f	1			0	n	n
2-Sep-99	4	e	5	3	2	1	n	n
3-Sep-99	1	c2	3	4	4			
3-Sep-99	2	b	5	3	1	0	n	n
3-Sep-99	3	a	4	2	2	1	n	y
3-Sep-99	4		2			0		n
4-Sep-99	1	g	5	4	1	0	n	n
4-Sep-99	2	c1	2	3	4	0	n	n
4-Sep-99	3	b	3	2	2	0	n	n
4-Sep-99	4	a	4			1	n	y
5-Sep-99	1	f	3			0	n	n
5-Sep-99	2	g	2		4	0	n	n
5-Sep-99	3	c2	4	3	1	0	n	n
5-Sep-99	4	b	5	4		0	n	n

Date	Data Lost	No of Notes Recorded	Paper Used?	Memory Used
28-Jul-99	y	28	y	0
28-Jul-99	n	140	n	301
28-Jul-99	n	0	y	0
28-Jul-99	n	28	n	198
29-Jul-99	n	29	n	205
29-Jul-99	n	52	n	280
29-Jul-99	n	43	n	212
29-Jul-99	n	36	n	217
29-Jul-99	n	24	n	208
30-Jul-99	n	56	n	216
30-Jul-99	n	89	n	289
30-Jul-99	n		y	
30-Jul-99	n	150	n	268
31-Jul-99	n	13	n	196
31-Jul-99	n	26	n	254
31-Jul-99	n	51	n	234
1-Aug-99	n	81	n	251
1-Aug-99	n		y	196
1-Aug-99	n	62	n	240
2-Aug-99	n			
2-Aug-99	n	76	n	292
2-Aug-99	n	20	n	
3-Aug-99	n	206	n	290
3-Aug-99	n	22	n	262
3-Aug-99	n	48	n	225
3-Aug-99	n	96	n	259
4-Aug-99	n	114	n	269
4-Aug-99	n	123	n	286
4-Aug-99	n	41	n	222
5-Aug-99				
5-Aug-99		136	n	
5-Aug-99	n		n	
6-Aug-99	n	121	n	281
6-Aug-99	n	42	y	213
6-Aug-99	n	100	n	
7-Aug-99	n	27	n	205
7-Aug-99	n	51	n	
25-Aug-99	n	53	n	236
25-Aug-99	n	128	n	299
25-Aug-99	n	152	n	
25-Aug-99	n	19	n	200
26-Aug-99	n	6	n	
26-Aug-99	n	51	n	
26-Aug-99	n	120	n	

Date	Data Lost	No of Notes Recorded	Paper Used?	Memory Used
27-Aug-99	n	29	n	
27-Aug-99	n	51	n	
27-Aug-99	n		n	
28-Aug-99	n	147	n	
28-Aug-99	n	11	n	
28-Aug-99	n	51	y	
29-Aug-99	n	10	n	
30-Aug-99	n	111	n	
30-Aug-99	n	8	n	
30-Aug-99	n	16	y	
31-Aug-99	n		n	
31-Aug-99	n	33	n	
31-Aug-99	n	63	n	
31-Aug-99	n	11	n	194
1-Sep-99				
1-Sep-99	n	129	n	
1-Sep-99	n	25	n	
1-Sep-99	n	148	n	
2-Sep-99	n	50	n	
2-Sep-99	n	51	n	
2-Sep-99	n	55	n	
2-Sep-99	y	22	n	
3-Sep-99			y	
3-Sep-99	n	10	n	n
3-Sep-99	n	53	n	
3-Sep-99	n		y	
4-Sep-99	n	139	n	
4-Sep-99	n	12	n	
4-Sep-99	n	9	n	
4-Sep-99	n	1	n	
5-Sep-99	n	125	n	
5-Sep-99	n	118	n	
5-Sep-99	n	12	n	
5-Sep-99	n	2	n	

Date Comments

28-Jul-99 PalmPilot had screen fault. Taking batteries out caused data loss.

28-Jul-99

28-Jul-99 PalmPilot lost templates because batteries were taken out over night (Ronda's email)

28-Jul-99 Loose connection on PalmPilot-GPS cable

29-Jul-99

29-Jul-99

29-Jul-99

29-Jul-99

29-Jul-99 screen still problematic

30-Jul-99

30-Jul-99

30-Jul-99 PalmPilot got stuck into one-handed mode

30-Jul-99

31-Jul-99

31-Jul-99 Flickering screen

31-Jul-99

1-Aug-99

1-Aug-99 Confusion over GPS UTM coords and PalmPilot Lat-lon coords

1-Aug-99

2-Aug-99

2-Aug-99

2-Aug-99

3-Aug-99

3-Aug-99

3-Aug-99

3-Aug-99

4-Aug-99

4-Aug-99

4-Aug-99 Veg plot + veg trees template would lock up after a while

5-Aug-99

5-Aug-99

5-Aug-99

6-Aug-99

6-Aug-99 gave up using PalmPilot because of resets needed every few trees

6-Aug-99

7-Aug-99

7-Aug-99

25-Aug-99

25-Aug-99

25-Aug-99

25-Aug-99

26-Aug-99

26-Aug-99

26-Aug-99

Date	Comments
27-Aug-99	
27-Aug-99	need paperclip.
27-Aug-99	need paperclip and battery recharge.
28-Aug-99	
28-Aug-99	
28-Aug-99	Batteries flat and spares also. 13 notes entered in field then the rest in the centre.
29-Aug-99	
30-Aug-99	
30-Aug-99	
30-Aug-99	
31-Aug-99	
31-Aug-99	
31-Aug-99	
31-Aug-99	
1-Sep-99	
1-Sep-99	
1-Sep-99	
1-Sep-99	Check batteries - low.
2-Sep-99	Recharge batteries.
2-Sep-99	
2-Sep-99	Batteries low.
2-Sep-99	Recharge batteries
3-Sep-99	Batteries out and programs lost.
3-Sep-99	
3-Sep-99	
3-Sep-99	Battery failed and had no spare.
4-Sep-99	
4-Sep-99	
4-Sep-99	
4-Sep-99	
5-Sep-99	
5-Sep-99	
5-Sep-99	
5-Sep-99	

Appendix I:
Kenyan Fieldwork Activities

Activity A: Tree Growth and Damage Measurement on a Plot

In this activity a team of two volunteers navigate to a pre-specified plot using the GPS receiver, where 50 or more *acacia drepanolobium* trees have been permanently tagged with numbered aluminium discs. Each tree's height, diameter and damage condition is then re-measured and recorded using the prototype tools. On returning to the research centre in the afternoon the volunteers download the data into a spreadsheet, and by comparing the collected data with the data for the same trees measured in 1998, calculate the annual growth rate and percentage damage for each height class.

Two stick-e note templates were created for this activity, one for recording the data about the plot and another used to record information about each tree, as illustrated in Table 30 and Table 31 respectively.

Name	Type	Description
Plot number	Number	The numeric ID of one of the 200 tree growth plots.
Location	Location	GPS derived location of the plot.
Comments	NotePad	Any comments about the plot; e.g. has it been burned recently.

Table 30. Growth-Plot stick-e note template.

Name	Type	Description
Tag number	Number	Tree tag number.
Height1	Number	Height of the tree from first position.
Height2	Number	Height of the tree from second position.
Height3	Number	Height of the tree from third position.
Diameter	Number	Tree diameter.
Damage	Picklist	General type of damage sustained by the tree: main stem, missing, dead, or none.

Name	Type	Description
		dead, or none.
Fresh damage	Boolean	Whether there is fresh damage to the tree or not.
Damage type	Picklist	Specific type of damage to the tree: main stem snapped off, tree pushed over, tree stepped on, main stem bitten off, tree leaning.
Damage height	Number	Height at which the damage was sustained.
Damager	Picklist	Cause of damage to the tree: elephant, rhino, giraffe, fire, natural causes, other.
Ants	Picklist	The species of ants living on the tree: rbb, rrb, rrr, bbb, brb, brr, rbr, bbr, or none.
Comments	NotePad	Any miscellaneous comments.

Table 31. Growth-Trees stick-e note template.

The growth-plot template is used only once when arriving at the plot, then for each tree a growth-trees stick-e note is recorded. Although the location of the plot is known in advance it is recorded again in a growth-plot note so that the project manager can verify that the volunteers attended to the correct plot.

Activity B: Rhino Identification

A team of two volunteers join the daily three-man armed rhino patrol at dawn to search for rhinos. The volunteers record the locations of all sightings of rhino and any rhino signs, such as footprints and middens (middens are the locations in which rhinos regularly defecate). Where there is positive identification, either by sighting or by prints, the name of the rhino (each rhino in the reserve has a unique name) and its location are recorded. If the rhino can be kept in view, the volunteers record on the PalmPilot all behavioural activities, such as walking, feeding, sleeping, etc. On their return the volunteers enter the location data into a spreadsheet for use by CALHOME (a computer program for calculating the home ranges of animals) and download any observational data onto a spreadsheet for later analysis. There are three separate templates, for creating rhino sightings, rhino signs, and rhino footprint stick-e notes respectively, as illustrated in Table 32, Table 33 and Table 34.

Name	Type	Description
Male name	Picklist	A picklist of the names of all the male rhinos in the reserve (Job, Jupiter, Kerkura, Loita, Otoro, Morani, Baraka, Rodney).

Name	Type	Description
Female name	Picklist	A picklist of the names of all the female rhinos in the reserve (Chema, Kilo, Mokora, Shemsha, Tamu, Waya, Saba, Tulivu, Carol, Ischerine).
Calf name	Picklist	A picklist of the names of all the calf rhinos in the reserve (no name, Robert, Tulia, Jama, Hifaldi).
Location	Location	GPS derived location of the sighting.
Activity	Picklist	Current activity of the rhino: walking, sleeping, feeding, defecating, spraying, scraping, standing, out of sight, wallowing, head rubbing, running, or drinking.
Comments	Notepad	Any miscellaneous comments, e.g. type of food being eaten.

Table 32. Rhino Sighting stick-e note template.

Name	Type	Description
Sign	Picklist	The type of rhino sign spotted: midden, bedding sight, footprints, dust bath, wallow, browse, or tree rubbing.
Location	Location	GPS derived location of the sign.
Comments	NotePad	Any miscellaneous comments.

Table 33. Rhino Signs stick-e note template.

Name	Type	Description
Location	Location	GPS derived location of the footprint.
A	Number	Footprint measurement.
B	Number	Footprint measurement.
C	Number	Footprint measurement.
D	Number	Footprint measurement.
E1	Number	Footprint measurement.
E2	Number	Footprint measurement.
Substrate	Picklist	Type of ground the footprint was found in: damp mud, dry mud, coarse soil, fine soil, sand.
Foot	Picklist	Which of the rhino's feet the print is from: front or rear.
Male name	Picklist	A picklist of the names of all the male rhinos in the reserve (Job, Jupiter, Kerkura, Loita, Otoro, Morani, Baraka, Rodney).
Female name	Picklist	A picklist of the names of all the female rhinos in the reserve (Chema, Kilo, Mokora, Shemsha, Tamu, Waya, Saba, Tulivu, Carol, Ischerine).
Calf name	Picklist	A picklist of the names of all the calf rhinos in the reserve (no name, Robert, Tulia, Jama, Hifaldi).

Name	Type	Description
		name, Robert, Tulia, Jama, Hifaldi).
Comments	NotePad	Any miscellaneous comments.

Table 34. Rhino Footprint stick-e note template.

Accurate location data is essential for this activity as the main use of the data is to analyse and identify which rhinos are where in the reserve, and what behaviours they exhibit in different areas. The prototype fieldwork tool ensures a reliable, accurate (relative to manual methods), and transcription-error-free method of collecting this location data.

Activity C1: Elephant Distribution by Eight Kilometre Transects

A team of two volunteers walks a specified five to eight kilometre transect and records all elephant dung piles within five metres of either side of the transect line (specified as a series of waypoints in the GPS receiver). GPS recordings are taken every half kilometre along the transect. On their return the volunteers input the data into a spreadsheet for processing and use by IDRISI (a GIS) to create an elephant distribution map. Only one template, as illustrated in Table 35, is used for this activity.

Name	Type	Description
Location	Location	GPS derived location of the sign.
No. of dung piles	Number	Number of dung piles counted in this leg of the transect.
End of transect?	Boolean	Checked if this is the last point on the transect.

Table 35. Dung stick-e note template.

The volunteers are often not able to follow the planned route of the transect due to dense vegetation blocking the path or encounters with dangerous animals (there are only so many dung piles one can find before encountering their creator!). Recording the GPS derived locations is therefore essential in ascertaining where exactly the volunteer did walk.

Each stick-e note records data for one half kilometre leg of the transect, during which time the dung count can be incremented using the MAUI controls each time an elephant dung pile is spotted (so leaving the volunteer's eyes free for spotting dung rather than operating the PalmPilot).

Activity C2: Tree Damage Sampling by Eight Kilometre Transect

A team of two volunteers walks a specified five to eight kilometre transect and records the location and condition of freshly damaged acacia drepanolobium trees within fifty metres of the transect line (which is specified as a series of waypoints in the GPS receiver). GPS readings are taken every half kilometre. After returning and downloading the data the volunteers calculate the mean percentage damaged trees for each section of the transect, and input the data into a spreadsheet as data points for using in creating a damage map in the IDRISI GIS. The single template used to collect the data is illustrated in Table 36.

Name	Type	Description
Location	Location	GPS derived location of the sign.
Damage	Boolean	If there is damaged tree here, or if the note is just being used to record the point along the transect.
Damager	Picklist	Cause of damage to the tree: elephant, rhino, giraffe, fire, natural causes, other.
Type of damage	Picklist	Specific type of damage to the tree: main stem snapped off, tree pushed over, tree stepped on, main stem bitten off, tree leaning.
Distance	Number	Distance from the tree to the transect line.
Bearing	Bearing	The direction of the tree from the transect line.
Height after	Number	The height of the tree after being damaged.
Height before	Number	The estimate height of the tree before being damaged.
Diameter	Number	The diameter of the tree at breast height.
Comments	Notepad	Any miscellaneous comments.
End of transect?	Boolean	Checked if this is the last point on the transect.

Table 36. Transect-Damage stick-e note template.

As with the dung-pile-counting transect, the ability to reliably record where the volunteer has walked and to allow for an eyes-free form of operation is essential. In fact, it is even more so as the volunteer has to look out for damaged trees within a one hundred metre area and to record the position of each one found.

Activity D: Tree Damage Sampling on Plots

A team of two volunteers carries out a half kilometre transect within one of the 25 plots with tagged trees in order to measure tree growth. The volunteers measure the height, diameter and damage status of all acacia drepanolobium trees within one metre either side of the transect line. The half kilometre is split into five lines each of one hundred metres length, the location of which is selected by the volunteers to give random detailed data on tree height distribution and damage status. The first tagged tree on the plot is the centre point of the study. On returning to the research centre, the data will be input to a spreadsheet and the percentage of trees damaged and killed in each height class calculated. Two templates are used to collect the data, one for describing the plot and the other for describing each tree, as illustrated in Table 37 and Table 38.

Name	Type	Description
Plot number	Number	The numeric ID of one of the plots.
Location	Location	GPS derived location of the plot.
Comments	NotePad	Any miscellaneous comments.

Table 37. Damage-Plot stick-e note template.

Name	Type	Description
Tree number	Number	Sequential number of the tree.
Height	Number	Height of the tree.
Diameter	Number	Diameter of the tree at breast height.
Damage	Picklist	Either main stem or no damage (damage to other parts of the tree should not be recorded).
Fresh damage?	Boolean	Recording if the damage is recent or not.
Damage type	Picklist	Specific type of damage to the tree: main stem snapped off, tree pushed over, tree stepped on, main stem bitten off, tree leaning.
Damage height	Number	Height at which the damage was sustained.
Damager	Picklist	Cause of damage to the tree: elephant, rhino, giraffe, fire, natural causes, other.
Ants	Picklist	The species of ants living on the tree: rbb, rrb, rrr, bbb, brb, brr, rbr, bbr, or none.

Table 38. Damage-Trees stick-e note template.

The use of the damage picklist in the damage-trees template is a good example of how the software prompts the user to enter the appropriate data whilst preventing them

from straying into performing redundant or incorrect tasks. It would be very easy to record any type of damage (as is done in the other activities) if the picklist did not restrict the volunteer's choices.

Activity E: Vegetation Sampling

A team of two volunteers navigates to a pre-specified location using the GPS receiver, where they select the nearest tree/shrub to the location and use it as the central point in marking out a circle of a thirty metre diameter. After recording specific details of the ground within the boundary of the circle they carry out a series of measurements on the central tree/shrub, and then repeat the measurements on the 9 nearest trees/shrubs to it. This data is downloaded to a spreadsheet and later used in the construction of a vegetation map of the reserve. Two templates are used to collect the data, one for describing the plot and the other for describing each tree, as illustrated in Table 39 and Table 40.

Name	Type	Description
Location	Location	GPS derived location of the plot.
% ground cover	Number	Percentage of ground covered by trees/shrubs.
Slope	Number	Angle of the slope (measured with an inclinometer).
Aspect	Bearing	Direction in which the slope faces.
Grass Ht 0	Number	Grass height at 0 metres across the circle.
Grass Ht 3	Number	Grass height at 3 metres across the circle.
Grass Ht 6	Number	Grass height at 6 metres across the circle.
Grass Ht 9	Number	Grass height at 9 metres across the circle.
Grass Ht 12	Number	Grass height at 12 metres across the circle.
Grass Ht 18	Number	Grass height at 18 metres across the circle.
Grass Ht 21	Number	Grass height at 21 metres across the circle.
Grass Ht 24	Number	Grass height at 24 metres across the circle.
Grass Ht 27	Number	Grass height at 27 metres across the circle.
Grass Ht 30	Number	Grass height at 30 metres across the circle.
Comments	Notepad	Any miscellaneous comments.

Table 39. Vegetation-Plot stick-e note template.

Name	Type	Description
Tree number	Number	The sequence number of the tree.
Distance	Number	The tree's distance from the central tree.
Bearing	Bearing	The direction of the tree from the central tree.
Species	Picklist	A selection of all the possible tree/shrub species.
Height	Number	Height of the tree.
Crown diameter	Number	Diameter of the widest point of the tree.
Lowest leaf height	Number	The height of the tree's lowest leaf.
Foliage shape	Picklist	A selection of general shapes of trees: cylinder, flat, hemisphere, sphere, cone (base up), or cone (base down). Used to calculate total bio mass of the tree.
Damage	Picklist	A selection of M, B, T, MB, BT, MT, or MBT, where M = Main stem, B = Branch, and T = Terminal shoot.
Bush	Boolean	If it is a bush or a tree.
No. of stems	Number	The number of stems if it is a bush.
Flowers	Picklist	The state of any flowers: buds, open, dead, buds + open, buds + dead, open +dead, all, or none.
Fruit	Picklist	The state of any fruit: unripe, ripe, rotten, ripe + rotten, unripe + ripe, unripe + rotten, all, or none.
Gum	Boolean	If the tree has gum coming out of its stems.
Ants	Picklist	The species of ants living on the tree: rbb, rrb, rrr, bbb, brb, brr, rbr, bbr, or none.

Table 40. Vegetation-Trees stick-e note template.

This activity generated the most data per stick-e note by far compared to all the other fieldwork activities. As a result it was in performing this activity in which the main bug of the prototype system would most frequently occur. This is worth describing briefly as it no doubt influenced the volunteer's perception of the tools. The bug would manifest itself in the form of an operating system "fatal error" message and the subsequent need to reset the device. Although inconvenient (and a little disturbing to the novice user!) the problem caused no loss of data except in the stick-e note currently being edited (which would have to be re-entered). After the reset the volunteers could continue with their task as per normal. We eventually traced the problem to a memory leak in part of the code handling the display of picklists.

Activity F: Rhino/Giraffe Behaviour in Morani's Compound

A team of two volunteers works in the 70 acre protected area which houses the black rhino called Morani. Morani can be safely approached and observed at close quarters when feeding on acacia and in the herb layer, giving valuable insights into how a rhino feeds. The data collected will be added to a rhino foraging database and compared with data collected in the reserve where rhino cannot be approached as closely. A single rhino-behaviour template is used to collect behavioural data on Morani, as illustrated in Table 41.

The behaviour is recorded by creating a new stick-e note for Morani's activity when he is first approached and then by creating a new stick-e note every time his activity changes. The prototype always automatically includes a time and date field in the stick-e note regardless of the template design, so there is no need to define such fields explicitly in stick-e note template. The time data is used to calculate a time budget of activities for the rhino.

Name	Type	Description
Rhino activity	Picklist	Current activity of the rhino: walking, sleeping, feeding, defecating, spraying, scraping, standing, out of sight, wallowing, head rubbing, running, or drinking.
Food type	Picklist	The general type of food Morani is eating (if his activity is feeding): acacia, scutia, herbs, grasses, euclea, or other.
Tree species	Picklist	If identifiable, the specific tree species that Morani is eating can be identified from an extensive list.
Herb species	Picklist	If identifiable, the specific herb species that Morani is eating can be identified from an extensive list.
Comments	Notepad	Any miscellaneous notes.

Table 41. Rhino Behaviour stick-e note template.

A wild male giraffe, called Dent, also lives in the same protected area and can be approached sufficiently closely to observe how a giraffe behaves. As with Morani, volunteers carry out a detailed study of Dent's behaviour, including how long the giraffe feeds on each tree. This data will be added to a giraffe foraging database at a later date. A single stick-e note template is used to record Dent's behaviour, as illustrated in Table 42.

Name	Type	Description
Activity	Picklist	Current activity of the giraffe: walking, standing-vigilent, standing-ruminating, walking-ruminating, running, feeding-acacia, feeding-other, interaction, other, or out of sight.

Name	Type	Description
New tree	Boolean	If the giraffe has started feeding on a new tree.
Feeding height	Number	The height at which the giraffe is feeding.
Bite rate start	Time	The time at which the giraffe starts biting the tree at that height.
Number of bites	Number	The number of bites taken at that height.
Bite rate finish	Time	The time at which the giraffe stops taking bites at that height.
Comments	Notepad	Any miscellaneous notes.

Table 42. Giraffe Behaviour stick-e note template.

The general behaviour is recorded in the same manner as with Morani, and the automatically provided time and date information is also used to build up a time budget for Dent's different behaviours. More detailed data can also be collected on Dent's feeding behaviour by completing a new stick-e note each time he engages in another feeding bout. This was described to the volunteers as an optional activity as there is an immense amount of data to collect in a very small space of time, as the giraffe can change trees, feeding heights, and take a number of bites all in a matter of seconds. The ability to automatically capture some timing data and to have the 'now' buttons to quickly update the other time fields made the activity at least feasible for a good proportion of the volunteers (those who were most comfortable with using the technology).

The rhino and giraffe behavioural observations were certainly the most demanding of the activities in terms of the speed and volumes of data recorded. However, the ease-of-use of the interface was demonstrated when one of Morani's guards, a local tracker with no previous computing experience whatsoever, started to help the volunteers by taking control of the PalmPilot and recording all of Morani's behaviour himself. In fact, he seemed to enjoy using the equipment so much that it has been difficult thereafter for the volunteers do any work with the PalmPilots themselves!

Activity G: Seedling Sampling on a Plot

A team of 2 volunteers performs quadrate sampling on one of the thirty plots that contain tagged trees. For this activity volunteers measure the height, diameter and damage status of all acacia drepanolobium seedlings and trees within one hundred 2m x

2m quadrates measured out with a thirty metre tape measure (see Figure 65). Every tree in the quadrate is measured and recorded, including any seedlings. The aim of this activity is to get a random sample of trees and seedlings so that the percentage height distribution can be calculated. Two templates are used to collect the data: one to record general information about the plot, illustrated in Table 43, and another used to record information about each individual tree or sapling, illustrated in Table 44.

Name	Type	Description
Plot number	Number	The numeric ID of one of the plots.
Location	Location	GPS derived location of the plot.
Comments	Notepad	Any miscellaneous comments.

Table 43. Seedlings-Plot stick-e note template.

Name	Type	Description
Quadrate number	Number	The numeric ID of the quadrate (see Figure 65).
Height	Number	Height of the tree of seedling.
Diameter	Number	Diameter of the tree or seedling main stem.
Damage	Picklist	Either main stem damage or none.
Comments	Notepad	Miscellaneous notes.

Table 44. Seedlings stick-e note template.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Figure 65. Sampling quadrate.

The summary data of previously recorded stick-e notes displayed in the StickePad main screen is essential for this activity. Without this reference of what data they have collected so far the volunteer can easily lose track of which quadrate of the one hundred quadrates they have progressed to.

Student Support Activities

In addition to the planned Earthwatch activities, a number of templates was also created to aid in the research of a number of MSc students also working at the reserve. In particular there were three students, studying baboons, lions, and giraffes respectively, who wanted to utilise the Earthwatch volunteers to record observations of their animals of interest if they came across them or any signs of them during their day's activity. The students could then collect the data gathered by each volunteer pair from the spreadsheet on the laptop computer at the end of the day. The necessary stick-e note templates only take a couple of minutes to construct, and provide a consistent, reliable, and efficient method of collecting extra data from the volunteers. The templates for recording information about lions, giraffes and baboons are illustrated in Table 45, Table 46 and Table 47.

The lion stick-e note template could be more straightforwardly constructed as a set of smaller templates for lion sightings, kills, faeces, and tracks. However, the template designers preferred to group all the information into one template so as not to

overwhelm the volunteers with a huge selection of templates to choose from when creating a new stick-e note.

Name	Type	Description
Location	Location	GPS derived location of the observation.
% tree cover	Number	Percentage of ground covered by shrubs and trees.
Species in sight	Notepad	List of species currently in the area.
Lion sighting?	Boolean	If a lion was sighted or not.
Adult males	Number	Number of adult males sighted.
Adult females	Number	Number of adult females sighted.
Sub-adults	Number	Number of sub-adults sighted.
Cubs	Number	Number of cubs sighted.
Lion kill?	Boolean	If a lion kill was discovered or not.
Species of prey	Picklist	Species of the kill: baboon, buffalo, bushbuck, eland, giraffe, grant's gazelle, hartebeest, impala, oryx, thompson's gazelle, warthog, waterbuck, zebra, or other.
Remains left	Notepad	Description of remains left.
Lion faeces?	Boolean	If there were lion faeces or not.
No. of faeces	Number	Quantity of faeces.
Substrate	Picklist	Type of ground surface: road or grass.
Lion tracks?	Boolean	If there were lion footprints or not.
Substrate	Picklist	Type of ground surface: road or grass.
Adult males	Number	Quantity of male tracks.
Adult females	Number	Quantity of female tracks.
Sub-adults	Number	Quantity of sub-adult tracks.
Cubs	Number	Quantity of cub tracks.
Comments	Notepad	Any miscellaneous comments.

Table 45. Lion stick-e note template.

Name	Type	Description
Location	Location	GPS derived location of the observation.
Bearing	Bearing	Direction of giraffe from the observation point.
Distance	Number	Distance of giraffe from the observation point.
Movement direction	Bearing	Direction in which the giraffe are moving.
Group size	Number	Total number of giraffe in the group.
Male adult	Number	Number of adult male giraffes in the group.
Female adult	Number	Number of adult female giraffes in the group.
Male sub-adult	Number	Number of sub-adult male giraffes in the group.
Female sub-adult	Number	Number of sub-adult female giraffes in the group.
Male juvenile	Number	Number of juvenile male giraffes in the group
Female juvenile	Number	Number of juvenile female giraffes in the group.
Comments	Notepad	Any miscellaneous comments.

Table 46. Giraffe Sighting stick-e note template.

Name	Type	Description
Location	Location	GPS derived location of the plot.
No of baboons	Number	Number of baboons in the troop spotted.
Comments	Notepad	Any miscellaneous comments.

Table 47. Baboon stick-e note template.

A major goal of all the student's studies was to estimate the population and distribution of their species. Having data collected by the 5 pairs of volunteers who were each day in different parts of the reserve was obviously a great boon to achieving that goal. The prototype fieldwork tools ensured that volunteers accurately recorded the location, time and date of the observations and that the rest of the data was collected in a consistent fashion.

Downloading the Data

At the end of each day's activity the data stored on the prototype was downloaded into the research centre's laptop computer. Although a quite straightforward process, this was probably the most complex activity to perform: especially so for the computing

novices who would have to get to grips with basics of using Microsoft Windows '98 in addition to the downloading process itself.

To download the data the volunteer would first log into the laptop computer, place the PalmPilot device into the hotsync cradle and then press the hotsync button. The device would then transfer all the stored stick-e notes to the laptop computer. Although this is all that needs to be done to save the data to the laptop, the project organisers also wanted the volunteers to import the data into an Excel spreadsheet. Indeed, it is likely that stick-e notes will often need to be compatible with a variety of different packages on the PC, depending on the particular application. To this end we provided a program called StickeStore on the laptop computer; StickeStore could export the stick-e notes to a tab-delimited text file, which could then be imported into the relevant application, Microsoft Excel in this case. Each pair had their own spreadsheet that they added their collected data to each day, and each spreadsheet contained a separate sub-sheet that corresponded to, and was named the same as, a stick-e note template.

Each volunteer was taken through the process of downloading on the first day in the field and then on subsequent days was given a step-by-step instruction sheet to help them carry out the process on their own, which the majority of the pairs successfully accomplished. Only one in five pairs required additional assistance after the second and third days. However, with the fieldwork computing tools not a single pair required additional help after the first day's training, other than the occasional query at the end of the day's work. Thus this software for downloading and exporting, which was not part of our research work, indirectly illustrated the research project's success in achieving ease of use.