Context-aware Applications: from the Laboratory to the Marketplace

Peter J. Brown, John D. Bovey and Xian Chen

Computing Lab., The University, Canterbury, Kent CT2 7NF, UK
{pjb,jdb,xc1}@ukc.ac.uk

ABSTRACT

Current hardware developments are making mobile computing increasingly attractive. An important class of mobile applications are *context-aware* applications: applications that change their behaviour according to the user's present context — their location, who they are with, what the time of day it is, ... This article is about software design for context-aware applications. Currently most such applications have been crafted by experts in research laboratories. Our aim is to factor out a simple class of context-aware applications and make the creation of these as easy as, say, creating WWW pages.

Introduction

Hand-held computing devices are becoming increasingly valuable to users. Firstly they are getting cheaper, more powerful and more ubiquitous, while battery life is getting slowly better. Secondly, there are emerging *de facto* standards in such areas as operating systems and inter-communication (e.g. the IrDA standard [1] for infra-red beaming). Thirdly, the world of mobile phones is moving towards the world of PDAs (devices like Psions, HP 300s and HP Omnigos) so that the user gets the benefit of both: on the one hand, mobile phones are becoming programmable, and, on the other hand, PDAs provide connectivity for phones. Lastly, and crucially for this paper, sensors are becoming available so that the hand-held device can keep track of its context; an example of this is Global Positioning System (GPS), which tells the user where they are. GPS receivers, including the antenna, are getting as small as a couple of inches square (see Figure 1). Indeed we will probably soon see products that combine a PDA and GPS system into a small integrated whole. Likewise a PDA might include a digital compass to give orientation, and environmental sensors to capture temperature, etc.

Because of these trends, there is increasing interest in *context-aware* applications: these are applications that change their behaviour according to the user's context. For example if the user is known to be at a certain location, then a context-aware application can present information relevant to that location. If the user has a communications link, this information might come from some central server, for example a server that monitored traffic on the roads. In general the information may be tailored to several aspects of the user's context: their location, the time of day, the season of the year, the temperature, ...

Existing context-aware applications

Context-aware applications have existed at least since the early nineties, and Schilit *et al* [2] published a good survey of them. Their survey describes several applications at the authors' home location of Xerox PARC. The applications are based on the *PARCTab* [3, 4] (see Figure 2), a device that is the size of a PDA, but which behaves very differently from a PDA. It is really a wireless dumb terminal that is continually connected, via an infra-red link, to a central system. It also incorporates active badge capabilities [5]. The PARCTab works in those rooms that have been specially wired up for it. The central system knows where each PARCTab is, and which PARCTabs are in the same room (see side panel). Assuming each PARCTab has a unique owner — and also assuming the PARCTab has not been borrowed or stolen from that owner — the central system knows who is with whom. In short, therefore, a lot of contextual information is available, and thus the PARCTab system provided a good basis for context-aware experimentation.

Several of the PARCTab applications created at Xerox PARC were concerned with presenting information. These included:

- presenting information about the room the user was in, e.g. when in the library they could be presented with information about the library. This information could either pop up automatically, or could be under the user's control (e.g. there could be a *This-room* button in the menu which, when selected by the user, gave information about the current room).
- helping the user find the most convenient local resource, e.g. the nearest printer. The PARCTab can present a menu showing all the available printers, and the distance of each from the user's current room.
- attaching a certain UNIX directory to a room; when any user enters the room all the files in the directory are shown. The directory is writeable by everybody, so anyone can post a note in a room by creating a file within its directory (in an office: "Back in 5 minutes", "Remember to turn the equipment off if you are the last to leave").

In general, context-aware applications are just concerned with triggering, and the information that is triggered is passed to some arbitrary program to process; the program can just be a simple application that pops up the information on the screen, as in some of the above examples, or it can be a program that performs some actions. The PARCTab had applications where the program was the UNIX shell, and the information passed to it when a certain context caused a triggering was a shell-script. The shell-script, which was run on a backbone UNIX system, might do such things as to log information or to bring up a window on a nearby static terminal.

Helping us to remember

The interest of the PARCTab work is the variety of small applications that were created. At the other end of the scale is an all-embracing project that is almost certainly the most ambitious context-aware application so far. This is the human memory prosthesis described by Lamming *et al* [6], and encompassed in their *Forget-me-not* system. This prosthesis is an artificial aid to help our memory, akin to an artificial limb to help walking. One device used by Forget-me-not is the PARCTab, and Figure 2 in fact shows Forget-me-not running: the user is recalling what happened at a certain time in the past, and the icons on the screen represent various people that were met and events that occurred.

The aim of the memory prosthesis project is to have a large number of sensors round the workplace, in order to capture as much as possible about the user's working life: what rooms they were in, who they were with, what communications they sent and received, and so on. This information is then available whenever the user wants to recall information ("What was the file I created [on a PDA, perhaps] while with Joan and Bill in the Conference Room about a month ago?").

Clearly the more information that is collected on a system such as this, the more there are Big Brother dangers. It is therefore vital to have an acceptable privacy policy, so that people can only see and use information that they could reasonably be expected to have. Privacy issues are complex, both in technical and social terms, and often lead to discussions where there is more heat than light. See [7] for a good discussion — primarily technical — of some of the issues. In fact most of the issues are not unique to context-aware applications: many people worry about the privacy of their e-mail, their use of web sites, the pattern and location of their computer use, etc; context-aware applications bring the issues to the fore by capturing and processing even more information about the user.

One facility that is used by the memory prosthesis is the *super-sensor*, as exemplified by Newman's [8] Pepys system. From low-level information about people's comings and goings, Pepys deduced higher level events such as meetings — crudely, a meeting can be recognised by several people coming to the same room at roughly the same time, though Pepys was cleverer than this. Thus Pepys acted as a super-sensor to recognise meetings, conference calls, seminars, and such activities. The general message is that context-aware applications can cover any sort of context, from that supplied by a simple physical sensor like a digital thermometer to that provided by a super-sensor that is an AI program that detects if a user is "busy", "interruptible", "in a meeting", etc., by collating information from lots of lower level sensors.

Further applications

The applications described so far represent two ends of a spectrum: the small and simple PARCTab ones, and the large and ambitious memory prosthesis project. There are many applications in between, covering fields such as guiding people in buildings [9], round a campus [10], or in field maintenance [10]. Some use special adaptations of WWW pages [11, 12].

Types of context-aware application

In the field of computer science, it would be good if we could classify applications into neat and separate compartments, but the world is not like that. Thus context-aware applications blur into other kinds of applications; indeed you could argue that *every* application that takes some account of the user is a context-aware application. In practice the adjective 'context-aware' is attached to applications that are *mainly* driven by the user's context. They tend to be mobile applications, because it is here that the user's context changes most rapidly, and it is here that there is the greatest need for context-aware behaviour. In fact, context-aware applications can themselves be divided (in a blurred way, of course) into *continuous* and *discrete* ones. In a continuous application, the information presented to the user changes all the time. An example would arise on a PDA with an accurate location sensor and a digital compass: an application running on this could continuously display an arrow on the screen to show what direction the user should go to reach a selected point — a point where a water pipe was reported as leaking, say. The arrow could be accompanied by a bar on the screen showing the current distance to the point. Continuous applications are challenging to implement. Fortunately most applications can more nearly be described as discrete: separate pieces of information are attached to separate contexts (rooms, time-ranges, being with certain people) and are displayed whenever that context is entered. Most of the applications we have described can be classified as discrete. It does not matter if the contexts in a discrete application overlap, e.g. they apply to overlapping areas: the key point is that the associated information is in discrete chunks.

Towards the marketplace

This article so far has argued that trends in hardware development are creating an increasingly favourable environment for context-aware applications, and has described some existing ones. In general, all these applications were created and used in research laboratories, and hand-crafted by highly qualified people. If the market opportunity for context-aware applications is to be taken, it must be made easier to create applications. That is where our own work comes in: our aim is to take an 'easy' subset of context-aware applications and to de-skill the creation of these. Specifically the aim is to make the creation of a context-aware application as easy as creating a WWW document. As will be seen, the ‘easy’ subset covers a large variety of applications, all towards the discrete end of the spectrum. Our aim is to be as general as possible, rather than, say, focussing just on applications where location is the key element of the context. The technology is based on what we call the *stick-e* note, which is the electronic equivalent of a Post-it note. Since our aim is openness and generality, a key part of the technology is that stick-e notes should be easy to exchange and to publish. They are therefore represented in a standard notation, SGML. (SGML is in fact a meta-notation: HTML, the notation used for web pages, is one instantiation of SGML, and is geared to representing such pages; our notation is another, and is geared to representing contexts. The syntax of each instantiation is controlled by a specification called a *DTD*, which defines all the SGML tags that can be used, and how they fit together.) A stick-e note embodies what a (discrete) context-aware application is: it consists of a context plus a *body*. Whenever the context is matched, the body is triggered. The DTD for stick-e notes specifies the tags that can be used to represent the contextual fields; these include:

- `<at>` for location.
- `<with>` for objects (people, equipment, etc.) that need to be with the user.
- `<during>` for time.
- `<facing>` for orientation.

The DTD is designed to make it easy for the author to define her own tags if she wants to augment the built-in ones; clearly this is a necessary feature if stick-e notes are to apply to all sorts of contexts, and hence all sorts of applications. Each stick-e tag for a field of a context has a *type*, just like a variable in a programming language has a type. The type for `<facing>`, for example, specifies a numeric value, or range of values, between 0 and 360. In fact ranges of values are used more often than single values: if you want to specify in a context that the user is facing South, you could do better to specify the range

```
<facing> 150..210
```

rather than

```
<facing> 180
```

because the latter would only be triggered if the user was facing exactly South.

The purpose of types, as in programming languages, is to help catch errors, in particular to make it likely that if the author makes a careless error, she will get an error message. Checking of types is implemented by the stick-e note software, when it loads a note.

Further details of this use of SGML tags can be found in other papers, particularly [14]; the aim here is just to give an overview.

The following is an example of a complete stick-e note:

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

(Some explanation: the tag `<required>` introduces the contextual fields — there are other alternatives to `<required>`, but we are not interested in such detail here; the co-ordinates following `<at>` give a rectangle that the user must lie within; in practice the co-ordinates are large numbers representing precise latitude and longitude, rather than the small numbers we show. The co-ordinates are normally generated automatically by the author actually being at the location concerned or pointing at the location on a map.) The above note would be triggered if the user was within the specified area, facing South in December.

Notes are collected together into stick-e *documents*. A document may, for example, represent a tourist guide to a certain city. Documents represent a package of notes, and are a useful unit for publishing and for defining what notes you want loaded, ready for triggering.

The software components needed

In order to implement a stick-e note system, you need:

(1) *SeTrigger*, the triggering module; at any one time, this has a number of stick-e documents loaded. If the context of any of the notes within these documents is matched by the user's *present context*, then the note is triggered. New notes can be added, or existing ones removed, at any time.

(2) the triggered notes are passed to a *SeShow* component. This can be any existing program: it could be a program that popped up notes as windows on the screen or, in the case where notes represent shell-scripts, it could be the UNIX shell.

(3) a set of *SeSensor* components, which periodically feed information to the triggering module in order to keep the user's present context up to date. These sensors communicate using a notation exactly similar to that used in notes: for example an electronic compass which wanted to record a reading of 172 would send the message

```
<facing> 172
```

In some cases all the SeSensor components will run on the user's PDA; in cases where the PDA has a communications link, however, the components may be distributed over several machines, as we shall see.

Labelled fields

Having explained the nature of stick-e notes, we will now concentrate on their essential feature: contexts. Before talking more generally about contexts, we will cover, in this Section, a small piece of infrastructure that is valuable in larger applications.

Some applications involve sensors covering many people or objects: for example a group of people (or, for that matter, vehicles) may have their location tracked. To cater for this, stick-e fields can have hierarchical labels: for example the position of a person who was an engineer called Sue could be represented by:

```
<at> engineer/sue = ...
```

The hierarchical rules for labels mean that if a note specifies `<at> engineer` then this will match any individual engineer, such as `engineer/sue`. This allows notes to be aimed at all of a selected group, in this case a group of engineers.

Enriching contexts

The beauty about the use of contexts is that, once you start using them, you find that they cover a wide variety of uses over and above values supplied by physical sensors. We have an application, for example, where share prices form part of the present context. (Since `<share>` is not a built-in tag, the author needed to define it as an extension for the DTD; its type is a numerical value from 0 upwards — a negative share price will give an error!) The 'sensor' in this case uses a broadcast share price service and feeds relevant values to the triggering module in a form such as:

```
<share> BP = 678
```

A corresponding note in the user's PDA might have the form:

```
<note>
 <required>
   <share> BP = 650..
<body>
 Consider selling BP.
```

Here the range `650..` means from 650 to infinity, i.e. any value greater than or equal to 650. This note would be triggered by the value of 678 used above.

As a further example, contexts can be enriched by imaginary fields set by the user. For example the author of a tourist application may provide some extra notes of special interest to architecture buffs: to do this the author can attach the extra field

```
<with> architect
```

The user's menu is then made to contain a set of possible companions (architect, historian, ...), and the user can select any or all of these if he wants to see specialized notes on various topics. In our implementation this menu is generated automatically according to which documents are loaded: thus the author does not have to bother with programming details. Such pretended companions mirror an idea used in some HyperCard stacks [15], where imaginary companions are used to help the user find their way selectively round a large body of information.

The idea of pretence turns out to be a surprisingly powerful aid to enhancing the value of contexts. For example our tourist application is based on a location sensor, but the user can pretend he is at a different location from the real one; he does this by pointing at a map — Figure 3 shows the user interface for this application. When he pretends he is in a certain context, it triggers all the notes for that context in the normal way. This allows the user, for example, to look ahead to see what notes would be triggered if he went to the location specified; he can use this as a basis for deciding whether to go there really. The user can even defy the old saying 'You can only be in one place at a time': he can pretend he is in a whole are, thus triggering all the notes for an area.

Pretence has turned out to be so useful that most applications allow the user to override any real sensor with a pretended value, thus allowing him to explore. Pretence is useful for testing: e.g. on a maintenance application the manager may like to pretend the temperature sensor shows a value below freezing point in order to see that appropriate notes are triggered.

The value of enriched contexts

Using rich and varied contexts, and thus allowing authors to tailor information more closely to their reader's needs, is a key to success for stick-e notes. Clearly, since stick-e notes are a general facility, a tourist guide constructed from stick-e notes may not be as good as a specially built tourist application; likewise a stick-e note share price pager may be worse than a specially designed share price pager. The gain however comes in the ease with which the stick-e author can incorporate extra contextual elements to make applications more general and more flexible.

Current status

We have implemented the system and produced several prototype applications, some of them albeit rudimentary 'proof of concept' demonstrations. SeTrigger has been written so that it can, if required, act as a server. Thus it can be used from other static computers and from mobile computers that have a communications link. SeTrigger runs on UNIX, on PCs and on HP200lx and HP Omnigo 700lx PDAs. (The HP 700lx is essentially a 200lx with a GSM phone attached.) The HP machines run DOS. In some cases we have used existing software to act as the SeShow module; in others, particularly the tourist application for PDAs that is shown in Figure 3, we had to write our own SeShow as we could not find any existing software that was suitable.

We have experimented with several sensors, including GPS, active badge, share prices, temperature (from an electronic thermometer attached to a building), etc. As you would expect, we have not found our location-sensing equipment ideal:

- In applications using PDAs with GPS the user needs to carry two separate devices, connected together with wires, and both have comparatively short battery lives. Matters are improving, however, and we hope to see an integrated PDA/GPS system soon.
- There is currently no suitable universal location system; GPS systems only work outside, and active badge systems only work in sites that have been suitably wired up. It is non-trivial to

translate between the co-ordinate systems of the two.

- The inaccuracies of GPS cause problems. In time the deliberate error (called SA) will be removed, and the situation will improve. There are also possibilities in using other satellite systems, and in using cellphones to detect location.
- We ordered some differential GPS equipment, in order to remove inaccuracies. In the UK there is a service that broadcasts GPS corrections using the signal of the *Classic FM* national radio station. When the equipment arrived, it came with a seven-foot antenna — some way away from a goal of having unobtrusive ‘wearable’ equipment.

Our feeling is, nevertheless, that in spite of these problems many of the most promising applications involve location-sensing, though there are certainly applications that exclusively involve attaching stick-e notes to contexts covering, say, environmental conditions or financial information.

Applications

Since we are interested in inter-communication for mobile people, our work has concentrated on the application of stick-e notes on PDAs, though many of the applications apply equally well to laptops and other larger computers. Several applications are described in [14] and we will not repeat the details here. We will, however, mention one application, which is not obvious and is perhaps conceptually amusing.

Stick-e notes can be used to cover paging requests. If I (pjb) receive a note — say over a phone link attached to my PDA — with the context:

```
<with> pjb
```

then it will immediately be triggered. I am always with pjb: it is something that I cannot get away from.

If I am in an environment where it is known who I am with (e.g. where active badges are used) then I can receive paging requests for other people; these are triggered when I meet the other person. For example the person might be a visitor, who is wearing an active badge identifying him as Visitor3 but who has no equipment for receiving paging messages. The note I receive might be

```
...
<with> Visitor3
<body>
Tell the visitor to report
to Bob before leaving.
```

This example is credible, but not very likely. It becomes, more interesting, however, if pieces of equipment or animals are used instead of Visitor3. There is active badge technology that tells someone when they are close to a certain piece of equipment, and it is also possible to have technology to identify small movable objects such as library books. Given this technology you could page a library book: the message might be "Jane wants to look at EP96 Proceedings book". The request does not go, of course, to the book itself, but to humans who might encounter the book. If one of these humans ‘meets’ the book, the paging message will come up on their screen.

Moving on to the case of animals, a number of animals now bear tags that uniquely identify that animal. Given this, a sensor in a milking shed can record which cow is being milked, or someone beside a valuable racehorse can know exactly which racehorse it is.

If animals have these tags, and a farm-worker or stable-hand has a PDA with a sensor that can detect individual animals, then a number of potential stick-e note applications become possible. These include ‘paging’ requests like the ones described above, but also include applications that do not require the PDA to have a communications link to the outside world. The worker may for example attach a stick-e note with the context of a certain animal with the message "check the swelling on my left rear leg". Whenever the worker meets the animal this note is triggered — when the animal had recovered it would, of course, need to be deleted. Such notes could be exchanged, perhaps by infra-red beaming, with other workers who met the animals.

Authorship

We have concentrated in this article on the form and usage of stick-e notes, but we should say a bit more about how an author sets about creating a set of notes.

The approach to authorship follows the simple Post-it note metaphor:

(1) You write a note on your PDA screen, using a keyboard, stylus or whatever other means is offered. As an example you might be on the outward leg of a hike, and your note, written on a PDA with a location sensor attached, says "Turned left here, by the pond".

(2) This is automatically stored as the body of a stick-e note, with the present context attached as its context. Thus if you are at a certain location, this is the context for the new note.

(3) Whenever you return to the same context, the note is triggered. Therefore, on the return leg of your walk you will be alerted to the point where you turned left on the way out.

At the end of the walk you might wish to save all the notes you made; these would form a stick-e document. You might give the document to a friend or even publish it on the web after you had refined it a bit.

The above approach applies equally well to our animal example. Here the context will be in the presence of a certain animal rather than at a location. We will take this to be the animal most recently detected by the sensor.

It is a truism that the hardest thing in designing a user interface is to keep it simple. The real world is full of complications, and the task of the interface designer is to factor out what is relevant to the user and to present this in a simple and unified way.

Our above description of the authorship process is indeed an over-simplification. Elsewhere in this article we have emphasized the value of rich contexts, and hence the present context may consist of several fields: at the very least it is likely to include the present time as well as the present location. If in step (2) above the time is attached to the new note as well as the location, it will only be re-triggered only if the user returns to the same point *at the same time*. This is only possible if time is wound back — actually not as absurd as it sounds, as stick-e notes can be used to replay past events (in this case the user pretends the time is some time in the past). Nevertheless in most applications the author would *not* want a time field to be attached to the newly-created notes. We therefore have a set of properties attached to each field, one of which says whether the field is to be used by default in newly created notes: in our example, location would, but time would not. The author can, if need be, override these defaults.

As a second over-simplification, we have assumed the walker has returned to exactly the same point on the way back. In practice he is unlikely to do so: therefore the matching process has a little play in it; in particular this is important if the location device is inaccurate.

In most applications, authorship in the field, i.e. the process described above, is the best approach. There is, however, nothing to stop authors creating stick-e documents by creating the SGML form of the note directly, just as many people create WWW pages by writing HTML.

Distributed processing

The user's PDA may be continuously or periodically connected to a backbone network via a wireless link such as a mobile phone. In these cases the stick-e software can be distributed between the backbone system and the PDA. For example:

- Some SeSensor components can run on the PDA and some on the backbone. Location, for example, can be detected on the PDA, but share prices, traffic information, and forecasts of future weather may come from the backbone.

- SeTrigger can run either on the PDA or the backbone or both. As an example of running on both:
  (1) A mobile user, who has a periodic connection to the backbone via a mobile phone attached to his PDA, phones a server on the backbone, giving it a context covering the whole area they are likely to be in during the next hour. The server triggers the notes that match this context, and down-loads them into SeTrigger on the user's PDA. (In this case it sends the entire notes, not just their bodies.)
  (2) The user relies on the SeTrigger running on his PDA, until he next feels he needs an update from the server.

With support from Vodaphone, we have experimented with the use of Short Message Service (SMS) to exploit a cellphone link to pass contexts and notes between an Omnigo 700lx PDA and the backbone. The advantage of SMS is that it is a store-and-forward system: thus it does not matter if the user's cellphone is temporarily switched off when, say, an SeSensor component running on the backbone sends a contextual update to the PDA. The disadvantage of SMS is captured in its name: 'short' actually means 160 characters, though there are ways round this.

Toys aren't us

All too often the output of researchers looks fine with small amounts of data ('toy' applications), but when used in the field with realistic amounts of data, the system reveals foundations of sand. We are conscious of these dangers, and have tried to avoid them. Obviously as the number of notes gets large there are potential problems of triggering speed [16] and bandwidth, but the biggest dangers, we feel, lie in areas associated with software engineering. It is well known that most of the cost of a program lies in its maintenance; it is less well known, but we think equally true, that most of the cost of documents also lies in maintenance. For example people who have created spaghetti structures of WWW pages are now beginning to see how much effort it costs to change and augment these structures as the underlying information changes. Stick-e notes are, perhaps, half way between a program and a document: certainly the contexts have to be as accurately specified as a program. There is little doubt that maintenance questions will be the limiting factor on how many notes it is viable to handle: would a tourist application of 10,000 notes be tractable? We have tried to anticipate the potential maintenance problems by adding features associated with programming languages: these include the use of types, declarations and testing facilities. Further details are given in [17].

Conclusions

There are many context-aware applications whose implementation will require the skills of highly-qualified systems programmers; such applications will be expensive to create. We hope, however, that this article has shown that there are also plenty of context-aware applications, which, given the stick-e infrastructure, can be created easily by people whose skills just lie in presenting information rather than in programming. We also hope we have provided a technology where exchange and publishing are easy, so that the huge benefits of these can be exploited. Finally, we have tried to create a technology which, if it does achieve the success we hope, has features that will allow it to scale up to cover the large potential applications that are ready to be exploited.

Acknowledgements

We would like to thank EPSRC for financial support for some of this work, and Xerox Cambridge for inspiration. We are also grateful for the constructive comments made by several people who read drafts of this paper, and to the anonymous referees.

REFERENCES

[1] IrDA Infra Red Data Association. Home page at URL http://www.irda.org. 1997.

[2] W. N. Schilit, N. Adams, and R. Want, "Context-aware computing applications", *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, IEEE Computer Society Press, Los Alamitos, Ca., pp. 85-90, Dec. 1994.

[3] N. Adams, R. Gold, W. N. Schilit, M. M. Tso, and R. Want, "An infrared network for mobile computers", *Proceedings of USENIX Symposium on Mobile Location-independent Computing*, Cambridge, Mass., pp. 41-52, 1993.

[4] R. Want, W. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis and M. Weiser, "An overview of the PARCTAB ubiquitous computing environment", *IEEE Personal Communications*, 2, 6, pp. 28-43, Dec. 1995.

[5] R. Want, A. Hopper, V. Falcao, and J. J. Gibbons, "The active badge location system", *ACM Transactions on Information Systems*, 10, 1, pp. 91-102, 1992.

[6] M. G. Lamming, P. J. Brown, K. Carter, M. Eldridge, M. Flynn, P. Robinson, and A. Sellen, "The design of a human memory prosthesis", *Computer Journal*, 37, 3, pp. 153-163, 1994.

[7] M. Spreitzer and M. Theimer, "Providing location information in a ubiquitous computing environment", *Proceedings of 14th ACM Symposium on Operating System Principles*, Asheville, N.C., SIGOPS, ACM, pp. 270-283, Dec. 1993.

[8] William M. Newman, Margery A. Eldridge and Michael G. Lamming, "Pepys: Generating Autobiographies by Automatic Tracking", *Proc. ECSCW ‘91*, Amsterdam, September 1991.

[9] S. Long, D. Aust, G. D. Abowd and C. Atkeson, "*Cyberguide: prototyping context-aware mobile applications*", GVU Center, Georgia Institute of Technology, 1996.

[10] A. Smailagic and D. P. Siewiorek, "Modalities of interaction with CMU wearable computers", *IEEE Personal Communications*, 3, 1, pp. 14-25, Feb. 1996.

[11] G. M. Voelker and B. N. Bershad, "Mobisaic: an information system for a mobile wireless computing environment", *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, IEEE Computer Society Press, Los Alamitos, Ca., pp. 185-190, Dec. 1994.

[12] M. F. Kaashoek, T. Pinckney and J. A. Tauber, "Dynamic documents: mobile wireless access to the WWW", *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, IEEE Computer Society Press, Los Alamitos, Ca., pp. 179-184, Dec. 1994.

[13] C. F. Goldfarb, *The SGML handbook*, Clarendon Press, Oxford, 1990.

[14] P. J. Brown, "The Stick-e Document: a framework for creating context-aware applications", *Proceedings of EP’96*, Palo Alto, Sept. 1996, published in *EP-odd*, 8, 2, pp. 259-272.

[15] T. Oren, G. Solomon, K. Kreitman and A. Don, "Guides: characterizing the interface", in Laurel, B. (Ed.), *The art of human computer interface design*, Addison-Wesley, Reading, Mass., pp. 367-381, 1990.

[16] J. D. Bovey, "The efficient retrieval of documents indexed by Boolean statements", submitted for publication, 1997.

[17] P. J. Brown, "Some software for the creation of context-aware applications", submitted for

publication, 1997.

Biographies

PETER J. BROWN received the B.A. degree in mathematics from Cambridge University in 1961, an M.Sc in mathematics from the University of North Carolina in 1963, and a Ph.D. in computer science from Cambridge University in 1968. He has worked at the University of Kent at Canterbury for most of the time since 1968, but has also worked for IBM, Stanford University, and the University of Canterbury in New Zealand. He is Professor of Computer Science at the University of Kent. His research interests have centred on the creation of novel software, and he has most recently worked in hypertext and in mobile computing. In collaboration with Office Workstations Ltd., he has won the annual award for best technical achievement from the British Computer Society.

JOHN D. BOVEY received a B.A. degree in mathematics from York University in 1968 and a Ph.D. in mathematics from York University in 1972. He has been a lecturer in Computer Science at the University of Kent at Canterbury since 1985. His research interests are in information retrieval and information display.

XIAN CHEN received the B.S. degree in information engineering from Xi'an University of Science and Technology of Electronics, P.R. China. From 1982, he worked in the area of signal processing in Xi'an Institute of Navigation Technology. In 1995 he received a Ph.D degree in Computing Science from Newcastle University, UK. He was a co-author of a book on parallel computing and automatic parallelization. At present he is a research fellow at the University of Kent at Canterbury, working in the area of personal information systems and mobile computing.

[[SIDE PANEL]]

The PARCTab

The PARCTab is a hand-held dumb terminal. Instead of having a wire connecting it to its host, it has a wireless infra-red connection. In order for the PARCTab to be used in a building, each room must be equipped with a transceiver — locally called the Death Star, which handles communication with all the PARCTabs in the room, and relays information to and from the host (actually there is a flexible distributed system rather than a single host). Thus there is a cellular network, where each cell is a room — large rooms are special cases. PARCTabs also act as active badges: this means that, even when they are not being used as dumb terminals, they periodically emit infra-red 'beacon' signals so that the Death Stars know where they are. Thus, if all is working well, the system will always know where all the PARCTab users in a building are.

[[FIGURE CAPTIONS]]

Figure 1. *the actual size of a GPS receiver, which includes an integrated antenna.*

Figure 2. *the PARCTab.*

Figure 3. *a tourist application on a PDA, showing a map and a note that has just popped up.*