# Facilitating the creation of context-aware applications

*P. J. Brown*

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

*ABSTRACT*

In context-aware applications, the computer automatically detects its context, and presents information to the user accordingly. This provides an interesting and promising area of mobile computing, since it can exploit mobility to create new forms of application.

If context-aware applications are to go into really widespread production use, two issues must be tackled: firstly, it must be easy for authors, who may have scant computer knowledge, to create applications; secondly, the implementation technology must scale up from toy applications to potentially huge ones.

This paper describes how the technology of *stick-e* sets out to tackle these issues.

**Index terms**—Context-aware applications, PDA, mobile computing, GPS, stick-e note

## 1. Introduction

The interest in mobile computing applications is partly in extending existing computer applications so that they can be used while on the move (e.g. e-mail, web browsing), and partly in developing completely new applications that exploit the mobile environment. An example of the latter is context-aware computing: the user carries a portable computing device, often a PDA, which is capable of detecting its context as it moves around, and the user is presented with information that relates to her current context.

In the simplest context-aware applications, the context may just consist of the user's location. This can be detected by such technologies as active badges [17], PARCTabs [1], GPS receivers, or even, with user interaction, by bar-code readers that are used to read bar-codes that are displayed around the workplace. Many existing applications are based on a tourism model: as the user moves round a city their current position is detected, and they are presented with information (on their PDA screen) about the sights of interest in their current position. This information can be augmented with route-finding information. The same basic approach applies not only to tourists in a city, but students finding their way round a campus [14], and, indoors, to visitors to museums or laboratories [8]. In some applications the location is augmented by the user's orientation (as might be detected by an electronic compass — perhaps mounted in their shoes so that it is pointing in the same direction as they are!), and this can be used to present information more selectively.

This paper is concerned with providing a framework to give authors a simple way of creating context-aware applications: creating an application becomes creating a document.

Context-aware applications can be classified into the following [18]:

(1) adapting the way information is extracted and presented.

(2) changing the user interface.

(3) changing the system configuration and the way commands work.

Clearly (2) and (3) require serious work by skilled systems programmers. We concentrate on (1), since it is here that there is a chance for a multitude of authors, who just possess imagination and some basic computing skills, to create a great range of applications.

From previous research involving electronic documents, specifically the creation of hypertext systems, we are well aware of the lessons from the success of the World Wide Web. A lot of us were building hypertext systems long before the web, and indeed these systems survive, but they have been swamped by the web. Some reasons for the success of the web are that its designers had the vision, which most of the rest of us lacked, to cover the following:

- information distributed over a wide community.
- a technology that scales, rather than one that works well for small applications, but breaks down with big ones. In any software the key to scalability is effective management of complexity.
- a technology that is applicable to a wide range of applications.
- a technology based on a simple underlying metaphor, which both authors and readers can readily understand.

Our aims are similar. In particular we want to cover a wide variety of *contextual element*s (location, orientation, time, companions, ... ), and allow these to be combined effectively. We also want to have a technology that scales up to large and diverse amounts of data: most current context-aware applications are prototypes of limited scale.

## 2. The stick-e document

Our approach is encompassed in the *stick-e* document. Details of this have been described elsewhere [1], [3], but in outline the approach is as follows. The underlying metaphor is the electronic Post-it note. In the simplest example of this the users carry a PDA with a location-sensing device attached. They write a note on their PDA when in a certain context (location); whenever they return to the same location the note pops up on the screen. (To be exact, it might pop up in the form of a symbol, similar to the symbol used to indicate that electronic mail has arrived.) Thus it is the exact electronic equivalent of a paper Post-it note that is displayed at some location in the workplace.

The basic Post-it metaphor (which is similar to that described by Schilit *et al* [13], except that the latter is based on a community of users who put notes in a shared file system) has been generalized to cover any form of contextual element. We call it a *stick-e note*. A stick-e note consists of a context together with the body of the note. A family of notes can be collected together into a *stick-e document*, and stick-e documents can be published or can be exchanged between individuals, perhaps by infra-red ‘beaming’ between PDAs, perhaps by e-mail, perhaps via a wireless network, perhaps on the web. An example of a stick-e document is a set of tourist notes covering a city. Stick-e documents are encoded in SGML to allow easy interchange. (The body within each note can be in any form, depending on what is being used to view the note. Thus if the viewer were based on a web browser, the form might be HTML.) The following is an example showing the SGML mark-up of a simple note:

```
<note>
  <required>
    <with> jdb
  <body>
   Sound out on being a
   partner in EPSRC bid.
</note>
```

The `required` tag introduces the context, which in this case is in the presence of the person `jdb`. Thus the note would pop up when any bearer of the note detected the presence of `jdb`.

## 3. Enriching the contexts

Many other contextual elements can be used in addition to location and orientation. A number of these have been explored in work at Rank Xerox Research Centre [6, 7] and at Xerox PARC [13, 18]; they include:

*Time:*
: time is readily available in any computing device. Context-aware applications can be boosted by a knowledge of the time: thus a stick-e note might apply to a garden location only in winter, or to a main street only between 9 and 5. (Indeed if we consider time as a contextual element, there is already plenty of context-aware software that runs on stationary computers: a clock that shows the current time is an example, though not, perhaps, a very interesting one.)

*Companions:*
: any location-sensing technology that has a central collecting point can detect when people are together. In particular, active badges can detect which people are in the same room. In terms of stick-e notes, a note with the context of "with Bill" will be triggered when the bearer of the note next meets Bill. In general, information about people's location can be provided by a Location Query Service [15] which covers all forms of object, and allows each person to protect their privacy.

*Proximity of equipment, animals, etc.:*
: Active badges can be attached to equipment, and maybe in the future it will be economic to attach such equipment to items such as library books. In addition animals sometimes carry devices that allow them to be identified. This is likely to use a completely different technology from active badges, and is not likely to use a central collection point; the application is, however, independent of the underlying technology. A stick-e note in the context of a particular library book may have the content "Jane wants to look at this". Any human carrying a PDA loaded with this note would, if the book was detected in their presence, see the note. Similarly a note in the context of a particular farm animal might say "Check that my left front hoof has properly healed". A stick-e document covering notes for a whole herd of animals could be carried by the farmhand currently on duty, and updated as necessary.

*The physical environment:*
: temperature, weather, etc.

*The computer environment:*
: for example moving from one directory to another can be treated as moving around a physical space.

*General use of sensors and public information sources:*
: examples are information on current speed of movement, current traffic problems, current share prices.

Overall, stick-e notes are a mechanism for triggering information, based on some combination of elements of the current context.

## 4. Pretence

All the above relate to the ‘real’ world. Perhaps the most exciting ways of extending the value of context-aware computing is to augment the real world with an imaginary world that the user pretends they are in. One way to do this is to allow the user to pretend they are in a different context from the one they are really in. Thus in a tourist application the user might pretend they are in a certain location in order to see what information relates to that location. To be specific, the user might have a map on their PDA screen, and might be able to point at the Buttermarket (thus pretending they are there) to see the information about the Buttermarket in order to judge whether it is worth visiting.

To carry this further, the user might be sitting in an armchair thousands of miles away, and be a surrogate visitor to the city. As a second example of this type of pretence, if I want to create a stick-e note that is to be triggered next time I meet Anne, then I can do this by first pretending I am with Anne, thus triggering any existing notes I have for Anne, and can then create a new note — it will relate to the current context, i.e. being with Anne; alternatively I might add to one of the existing notes.

The more interesting type of pretence, however, goes beyond this, in introducing new elements of the environment that are entirely pretended. Thus there might be pretended companions, pretended buildings, etc. This facilitates the approach to managing complexity that Apple used when they designed guided tours of HyperCard stacks [11]. With their HyperCard stacks covering American history, Apple allowed the user to select imaginary companions (e.g. a historian, a trapper, a soldier) who would guide the user on a tour of part of the material; the purpose of this mechanism was to help the user to explore a large and complex amount of information. In terms of stick-e notes, the same can be done by adding the appropriate tour guide(s) as extra companions in the context of the appropriate notes (e.g. "at the Buttermarket with a historian" would be the context for a note about the detailed history of the Buttermarket). To carry this example further, there might also be pretended times: the user could pretend to be in the seventeenth century, and see the notes that had the seventeenth century in their context. All this pretence is, of course, under the control of the user: no sensing device will detect an imaginary companion (unless a detector of the smell of alcohol could deduce that the user was ‘in the company of Bacchus’), nor an imaginary time. Instead the user must explicitly say that they are taking up/discarding a pretended companion, or moving into or out of a pretended time.

The aim of the above application is to provide a simple user interface covering a potentially large amount of information; our second application is one where the simple user interface is equally important, though the amount of data may be much less. In this second application area, imaginary companions and/or locations are used in stick-e documents to create a generalised ‘To do list’ application. As one example, the user can create an imaginary location ‘Supermarket’ (assuming that no real sensor will detect when the user is in a supermarket): whenever they want to add something to the shopping list they can pretend they are in the Supermarket and post the note. When they are in a real supermarket, they can say they are there and trigger all the relevant notes. As a second example, using an imaginary companion rather than a location, a user who is a stamp collector can create an imaginary companion Stanley Gibbons; they can then post and view notes relating to their hobby by pretending they are in the company of Stanley Gibbons.

None of the individual examples quoted in this Section is essentially new, but we hope that collectively they show that stick-e notes can act as a simple metaphor to bring together and to generalise a diverse set of applications.

## 5. Special values of contextual elements

We now move a stage further with the idea of pretence; one side-effect will be breaking away from the real-world constraint embodied in the statement: "I cannot be in two places at once".

Generally each contextual element has a specific value: thus a location will be a set of internal coordinates. Extending the idea of pretence, stick-e notes also allow *special values*, such as "any", "none", or "unset". If the user pretends to be in "any" location, he is everywhere; this is a way of triggering all the current notes that are based on location. Thus "any" provides an overview facility, though, if there is a large number of current notes its effect can be overwhelming. (As an intermediate stage, a user can pretend he is in the whole of a area, rather than at a single point.) The special value "unset" (which also covers "not working") can be used to provide notes that provide information in situations where the expected hardware is not there — this value therefore can relate to the real world, rather than an imaginary one.

### 6. Implementation: the software components

Having covered some of the ideas behind stick-e notes, we now move to implementation. A general stick-e note prototype has been implemented, and many applications created — though mostly in outline form at this time. These applications include touring (both outside and inside), share-price paging, "to do" lists, a diary, and launching computer programs. Some of these use simulated location-sensing devices. Details of the software components used to implement stick-e notes are given in [2], but the most fundamental components are:

- *SETRIGGER*, which deals with the triggering of notes. The notes that are currently loaded into SETRIGGER are called the *primed* notes. The task of SETRIGGER is to detect when the context of any of its primed notes is entered, and, when this happens, to trigger the note. Thus a note whose context is the location of the Westgate will be triggered when the user nears the Westgate (locations are normally set to a small area surrounding the object in question; this is all the more necessary if the location-sensing technology can be inaccurate, e.g. GPS). In some applications, such as tourist ones, the set of primed notes will remain constant over a long period of usage, whereas in others, such as paging applications, the set of primed notes may be changing all the time. In any application where the user is an author, the set of primed notes will, of course, be changing as the author creates new notes.
- *SESHOW* is a component that receives the notes triggered by SETRIGGER, and presents them to the user. In principle different primed notes can relate to different SESHOW components — e.g. paging notes may have a special SESHOW of their own — but, in our current prototype implementation, when SETRIGGER is started it is told the name of a single SESHOW component, which is then used for all triggered notes. Thus, when SETRIGGER triggers a note, it passes a message to SESHOW, giving it the body of the triggered note.
- Lastly, there are a number of *SESETENV* components, which tell SETRIGGER about the current context. (The word SETENV, incidentally, comes from the UNIX `setenv` command, which has some parallels with SESETENV.) SESETENV components are attached to physical sensing devices, such as GPS receivers, or, for pretended contexts, are components that relay to SETRIGGER the pretended settings that the user makes.

These components may be packaged and distributed (see below) in different ways, depending on the hardware and the application. In many applications, the SESETENV components used to set the pretended context are packaged with SESHOW, so that the user sees what appears to be a single application. The user may, for example, see:

- a menu to cover certain settings of the pretended context.
- a map: the map shows the user's current location; by pointing at the map, however, the user can pretend they are in a different location.
- the currently triggered notes.

An example is shown in Figure 1.

*Figure 1: layout of a stick-e application*

The 'user' of SESHOW need not be a person: it could, for example, be a program. In one of our prototype applications, for example, SESHOW is the UNIX shell and the body of each stick-e note is a UNIX shell script;

when a note is triggered the shell script is passed to the shell to be executed. In this case stick-e notes simulate the Context-Triggered Actions described in [13]. In another example, to be discussed below, SESHOW is a further invocation of SETRIGGER, to do a second layer of triggering.

## 7. Implementation: distributing the components

In the simplest case, context-aware computing applies to a PDA which has no connection with the outside world, other than, perhaps, the occasional down-loading and up-loading from a stationary computer on the wired backbone network. Research in mobile computing has not traditionally been interested in this simple case — even though it is mobile and involves computing — but context-aware applications have the merit of making this simple case more interesting.

In a slightly more connected case, the PDA may be in periodic contact with the outside world: a travelling salesman may use a mobile phone to update his data — in our case stick-e notes — for new contacts that have arisen over the previous hour or so.

In most applications where there is a rich variety of contextual elements, however, the PDA will be in near-continuous wireless communication with a backbone system. Some of the context will be derived from the user's PDA, e.g. their position, and some from the background network, e.g. the weather. In terms of new notes, some might be created by the user on their PDA, and some, e.g. paging requests, might originate from the background network.

SETRIGGER can be greedy of computing power if there are a large number of primed notes; thus it is a candidate for running on a stationary computer rather than a mobile one.

Overall, therefore, the implementation of stick-e notes can benefit from a flexible distributed environment, as provided, for example, by the InfoPad [9], by Service Proxies [4] or by Odyssey [12]. Such matters are not the main focus of this paper, but we will discuss just one aspect: the use of agents.

In many mobile applications there is a vast amount of potentially useful information available, perhaps coming from broadcast sources; the user's PDA will only have the capacity to hold a small proportion of this. In such cases there is a need for a software agent to filter out what is needed. Often, for reasons of capacity and/or power consumption, the agent will run on a separate computer from the user's. The agent's knowledge of the user's context may be uncertain [5], depending on how well the agent is connected: (a) to the user, and (b) to its information sources. If the agent is only in touch with the user periodically (e.g. information is downloaded to the user's PDA once an hour or even once a day, or the information source is only available once an hour or even once a day) the agent needs to anticipate what contexts the user will enter before the next update. The agent therefore needs an *anticipated context*, which is the union of the likely contexts that the user may enter before the next update. If the information is location-based, for example, the anticipated context may cover an area of several square miles around the user's present location.

If the information source consists of stick-e notes, or can readily be converted to this form (i.e. the context can be readily separated from the information content), then the agent can just be a copy of SETRIGGER. This SETRIGGER agent just matches the anticipated context against all the contexts on the stick-e notes coming from its information sources, and triggers those that match; these triggered notes are then passed to the user's SETRIGGER (thus SESHOW to the agent SETRIGGER is actually another copy of SETRIGGER) so that they can be triggered if the user does indeed enter the various parts of the anticipated context.

Thus we have used the same technology, as embodied in SETRIGGER, to cover both the agent and the user's triggering. The result is a considerable gain in simplicity, and in flexibility on how systems can be configured.

## 8. Mixing pretence and reality

We now move on to an issue that is at one level an implementation feature, since simple-minded authors can ignore it, but at a higher level gives power and flexibility to ambitious authors.

The concept of pretended contexts, and their reasonably seamless interface with real contexts, is one of the main sources of power for stick-e notes. Behind the scenes, however, there are differences between pretence and reality which some authors, at least, need to be aware of. In particular the matching process by which notes are triggered is affected.

The way SETRIGGER works, when doing its continual matching work, is to take the *present context* (i.e. what the user's current context really is, or is pretended to be) and try to match it against the context of each of the primed notes. Those notes that match are triggered. Both the present context and the context on a note consist of the ANDing together of several contextual elements; for example a context may say "at a certain location AND with a certain person". (There are also facilities for ORing, etc., but that is not of relevance here.)

Consider some examples of matching. In the first example the user's computer has sensing devices that set a wide range of contextual elements within the present context: location, orientation, temperature, barometric pressure, humidity, etc. If a given stick-e note has a context that just specifies a location and an orientation, then it will match if these match the present location and orientation of the user; the present values of temperature, pressure and humidity are immaterial. Similarly if the note's context just specifies a humidity, the present values of location, etc., are immaterial, and the match is solely determined by whether the note's humidity matches the present humidity.

As a second example, assume the user has said, as a pretended part of the present context, that he is with Tourguide A, and that this means that he only wants to see notes that have "with Tourguide A" in their context, i.e. that he wants to confine himself to a particular tour. This is different from the previous example in that a matching note *must* contain the contextual element "with Tourguide A": a note that just specifies a location will not match, irrespective of whether it matches the present location — in the user's terms it is not relevant to his tour and is therefore ignored.

To cater for the difference that these two examples show, a context is, in fact, divided into two parts: a *required* part, which must be matched, and a *casual* part, which is matched if it is needed. This applies both to the contexts on every stick-e note and to the user's present context, though the contexts on stick-e notes often just consist of a required part. Normally all the parts of the present context that are automatically set will be in its casual context, while parts set explicitly by the user *may* be in its required part. In our examples, therefore, the location, etc., would be casual, but "with Tourguide A" would be required.

The reason why we use the word "may" above is illustrated by the third example, a variant on the second. Here the user has pretended he is with an Architect, with the intention of seeing all the notes that have "with Architect" in their context — these will be the notes for architecture buffs — *in addition to* the notes they would see anyway. In this case the Architect is a casual part of the present context, just like the location, etc.

To make the next point we will follow the Victorian practice, unusual in IEEE publications, of flattering the reader. Even you, the intelligent reader, a world expert in mobile computing, ... , may have had trouble grasping the significance of required and casual contexts, — due, of course, to the inadequate explanation given by this humble author. It is essential, therefore, that the user of stick-e notes does not need to grasp it: thus the distinction must be presented in terms of some application-oriented metaphor, exclusive companions and extra companions perhaps, or even not presented at all (default assumptions being made). In terms of authors of stick-e notes, they can create a lot of applications using the rule that every note's context is always a required context. To create more sophisticated stick-e documents, however, a deeper knowledge is needed, as our fourth and last example illustrates. This shows a case where a note needs to have a casual context too.

In this last example the author has a note about St. Augustine's Abbey, which is seen by a user with Tourguide A, but is also seen by any user who is located at the Abbey. Although in most cases the context of a note is solely a required context, for this note, "With Tourguide A" is not required, but is casual.

To complete the explanation, this is the matching process between the present context and a note: the required context on the note must be a subset of the total present context (required plus casual) and the required part of the present context must match the total context of the note.

In our experience so far, covering a variety of prototype applications, this distinction between required and casual has worked well.

**9. Managing complexity**

Like any technology, if stick-e notes are to succeed the technology must scale. The biggest challenge is likely to be maintenance. The author must be able to look after thousands of stick-e notes, changing them and adding to them over a long time period.

At the time of writing, the working context-aware applications reported in the literature operate in terms of tens or hundreds rather than thousands, and most of them are used in research environments. In the future we expect to see production applications on a much larger scale.

Stick-e notes offer *some* aids to attacking complexity. One popular weapon against complexity is the use of structuring: stick-e technology offers a somewhat feeble aid here in that notes can be collected together into separate documents.

A second and more powerful aid is concerned with error checking. At the head of each stick-e document the author can declare what contextual values are used in the notes within that document. For example the declaration might say that:

- all locations used in the document must come within a certain area.
- the only possible companions are X, Y and Z.
- no other contextual elements can be used.

This declaration takes the form of a context, and its SGML encoding looks just like the context on a note. It is called the *world context*. If a world context exists, then SETRIGGER, as it loads each note ready for priming, checks that its context matches the world context, and reports an error if not. In a nutshell, world contexts are invaluable in saying what contexts are used in a document, and providing a verification process.

A third aid, supported by the current stick-e implementation, is that an option is provided whereby SETRIGGER explains why each match is made. This option was originally provided for users — if they entered a crowded room and a note was triggered, they might like to know why — but it also has proved useful in helping authors to find bugs in the contexts they have specified for notes. Software designers are familiar with the phenomenon that some facilities they provide never quite come off, whereas others turn out to have uses way beyond the original design aim. The explanation facility in SETRIGGER has proved to be a fine example of the latter, and has turned out to have a third advantage in addition to the two just quoted: it has proved useful in reducing the size of stick-e documents. One application, for example, uses stick-e notes to act as a share-price pager: the context for a note might be "Shell > 800", which means that the user wants to be alerted if the price of Shell shares passes 800. SETRIGGER has a feed of current share prices, and triggers the notes whose context is matched. The author for this application set the option to explain all matches, and the explanation for our Shell note might be "Present context of Shell = 810 matches the note's context: Shell > 800". This explanation is all the user needs (in other words the reason why the note was triggered is itself the message the user needs): the bodies of all the notes in this application are null, thus saving a lot of potentially error-prone authorship effort.

A fourth aid to managing complexity is the use of pretence by authors in order to investigate the nature of a collection of notes. (Maybe the notes were originally written by a different author, or by the same author some months previously.) As an example assume that a contextual element in a stick-e document is temperature: some notes are triggered at 0°F, some at −10°F, etc. The author can get a feel for the document by successively pretending that the temperature is each of the key values, and seeing what is triggered. (The key values might even be declared in the world context as the only possible values.) Better still, the author can construct a semi-automatic testing process using a *tour*, which successively goes through a prescribed sequence of contexts in order that the author can check what notes are triggered at each stage. The tour is the last aid to managing complexity, and perhaps the most useful aid of all. It is described in the next Section.

## 10. Sequential tours

There are many applications of context-aware computing where the idea of sequence is important. This is most obvious on a sequential tour, where the user goes from one location to another. (Often such tours are circular, i.e. the first location is the same as the last.) The concept of sequence can, however, cover any context, such as our 'tour' of the stick-e notes that are triggered when a temperature within the present context achieves certain values. In the paper so far we have not mentioned sequencing. We have discussed tours — see previous material on HyperCard stacks — but, as we have described them so far, there is no built-in sequence (though the text within the bodies of individual notes may well suggest where the user should go next). From now on we will use the word "tour" in a more general way to mean a sequential tour relating to any context.

When the design of tours within stick-e notes was being considered, there were, of course, many requirements, but the most important were:

- the user should be provided with commands such as Next, Previous, and Restart, in order to go through the tour. Commands such as Nearest (meaning find the nearest — by some metric — tour point to the present context) should also be possible.
- Tours might be real, in the sense that the user is travelling around as she takes the tour, or even in a room where the temperature is successively reduced to 0°F, −10°F, etc; alternatively the tour can be pretended, a preferred option in the temperature case.
- If the author desires, it should be possible to add a tour on top of an existing stick-e document, which we call the *underlying* document. For example if an underlying document contains a set of stick-e notes about a city, an author — perhaps independent of the author(s) of the underlying document — should be able to create a new stick-e document, which, if loaded together with the underlying document, provides one or more tours of the city. The existence of a tour should not necessitate any changes in the underlying document.
- Following on from the previous point, the existence of tours should not affect users who are not taking the tours.
- Users should be free to join and leave tours without any hiccup, though, as in real life, if the user tries to take two tours at once there is likely to be confusion.
- The tour should be largely independent of changes in the underlying document.

> Thus if a tour visits Canterbury Cathedral and new notes for the cathedral are added to the underlying document, then the tour should see these too. (If the location of Canterbury Cathedral were to be moved, there might be a problem; however this last happened in 1067 AD; if, on the other hand, the tour covered, say, exhibits in a museum the problem would be more serious.)

The design to meet these needs is as follows. A tour is represented as a sequence of *tournotes*, each of which is a special form of stick-e note that has two contexts: the normal context and the *setting* context. In the simplest case the tournotes corresponding to a tour are written as a sequence, one after the other, and each has the same normal context, which often consists just of the imaginary companion who is the guide for the tour. We will assume here that each of the tournotes relates to Tourguide B. The user who wants to take this tour therefore places "with Tourguide B" in their present context. When the user selects a Next command, this is what happens: the present context is matched against the normal context of each tournote in turn, starting at the one last visited, or, failing this, at the beginning. When a tournote is matched, its setting context is used to re-set the present context, and this new present context is used to trigger notes in the normal way. Thus the setting context might be the location of the next site to be visited, or the next temperature to be pretended, ... ; in each case, all the notes for the new setting would be triggered.

The basic tournote mechanism can be exploited in various ways by using casual and required contexts, and by using different combinations of contextual elements.

Obviously this facility for tournotes itself adds complexity to the original simple idea for stick-e notes. It can, however, be presented to authors as a few simple rules: to create a tour, specify an imaginary tourguide and create a sequence of tournotes with this tourguide as the imaginary companion (the best default is as a casual one), and with the points on the tour as the setting contexts.

In practice the tournote facility has, reasonably satisfactorily, met the above requirements for tours.

## 11. Extensions

Two possible extensions could improve the capability of stick-e notes to cover a wider class of applications. Each is based on dynamic environment variables, like those used in [16]. Firstly contexts can be made more general by allowing the author to run a program to set a contextual element: for example the share price pager might want to run a program that compared a current price with a past price, and only triggered if the increase was greater than the increase in some index. When this happened the program could set some environment variable which is part of the current context; the stick-e note would use the setting of this environment as a condition for triggering.

Secondly, the body itself can have values of these environment variables inserted: for example part of the body might be "You are at X" or "Load file ...X", where X is filled in according to the current location.

Such extensions are in principle easy to make to the current stick-e note implementation. The real challenge is to make them in such a way that ease of authorship is not prejudiced.

## 12. Applicability of stick-e notes

Finally, we will try to classify the type of applications for which stick-e notes are suitable. Clearly the stick-e note technology might not represent the whole application, but might be used to implement one module within it. For example the application might be a general information system, and stick-e notes could provide the context-dependent parts of this. As an alternative the application could be a context-sensitive user interface that was driven by a table specifying the current interface, and the stick-e notes could determine which table to use.

Stick-e notes are only applicable for application modules where the following hold:

(1) the contexts to be used are discrete (though they can certainly overlap). Thus stick-e notes are suitable for collecting and for displaying data about each of a set of wheatfields, but not for recording or showing a continuously changing reading as the user moves across a wheatfield. (This is a consequence of the stick-e mechanism being based simply on triggering.)

(2) the notes themselves do not change continuously. With stick-e notes a predefined context is passed to SETRIGGER, and when a note is triggered its predefined body is passed to SESHOW. A degree of dynamic behaviour is, however, possible. Firstly, SESHOW can dynamicly obtain information: for example SESHOW could a module to show the current weather forecast, and the predefined body could be the address of a place that SESHOW should use to obtain the current forecast — the predefined body would not be the forecast itself. Secondly the author can edit a note (this counts as deleting an existing notes and creating a new one), and, more generally, some or all of SETRIGGER's existing notes can be cancelled and/or new ones loaded. These facilities are, however, aimed at incremental change rather than continuous change.

(3) the whole module can be described in terms of IF-THEN relations, which is essentially what stick-e notes are.

(4) one or both of the following is a requirement: simplicity of authorship and usage; the ability to publish and exchange notes in a standard form (SGML).

In essence, as we have emphasized throughout, stick-e notes are a simple metaphor to cover simple applications.

An example may help to illustrate the boundaries. This example covers a case where stick-e notes are used as a module to help implement mobile systems. Specifically, stick-e notes can be used to define the contexts and consequent actions in an application-aware environment [10], [4]. In an application-aware environment, the application is re-configured as the circumstances of the mobile client change; stick-e notes can potentially be used to describe these circumstances. By and large this meets the conditions (1) to (4) above. An individual note can consist of

(a) a hardware configuration as its context.

(b) the consequent action as its body; clearly in this application the SESHOW component is a program, and the body of each stick-e note is a command to be passed to the program, telling it to make some adaptation.

SETRIGGER is used to monitor configuration changes (e.g. the nature of the wireless network used), and when a change in the configuration causes a new note to be triggered, the corresponding body is passed as an action to an SESHOW component which effects the necessary changes.

Nevertheless this is not one of the most promising applications of stick-e notes. The essential issue is that (4) above represents the basic advantage of stick-e notes, but in this application the advantage is relatively small. The author is likely to be a systems programmer and the user is not a person at all but a program. Having a standard form for exchange is perhaps not a big advantage. On the other hand (1) to (3) might turn out to be substantial constraints, especially at the detailed level. Furthermore the benefits of stick-e notes as a way of managing a complex amount of context-related information are perhaps marginal in application-aware applications, where the amount of information may be relatively small.

## 13. Conclusions

To conclude we will move away from the above rather negative example, and will be more positive. The technology described in this paper, the stick-e note, has, we hope, been shown to have the following benefits:

- it is a simple metaphor, easy for authors and users to understand. Authorship is no harder than it is to create a web page in HTML.
- it applies to many existing and potential context-aware applications.
- it brings together pretence and reality in a way that enhances the power of both.
- it makes a start in tackling the problems of managing complexity, and thus offering a scalable technology. Since there is always a price to pay, the efforts to tackle complexity themselves add complexity. Early results indicate that the balance comes out substantially positive.

The real measure of success, however, is not claims such as the above, but whether the technology really helps to move context-aware applications from the research laboratory into the outside world, so that there is a new class of widely-used mobile applications.

**REFERENCES**


[1] 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.

[2] P. J. Brown, "The Stick-e Document: a framework for creating context-aware applications", *Proceedings of EP'96*, Palo Alto, Sept. 1996.

[3] P. J. Brown, "The electronic Post-it note: a model for mobile computing applications", *Proceedings of the IEE Colloquium on Developments in Personal Information Systems*, London, pp. 6/1-6/3, June 1995.

[4] A. Hokimoto, K. Kurihara and T. Nakajima, "*An approach for constructing mobile applications using service proxies*", Japan Advanced Institute of Science and Technology, Ishikawa, 1996.

[5] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: challenges in data management", *CACM*, 37, 10, pp. 18-28, Oct. 1994.

[6] M .G Lamming and W. M. Newman, W. M., "Activity-based information retrieval: technology in support of personal memory", in F. H. Vogt (Ed.) *Personal Computers and Intelligent Systems*, (Proceedings of Information Processing 92, Vol. III), Elsevier North-Holland, Amsterdam, 1992.

[7] 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.

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

[9] S. Narayanaswamy, S. Seshan *et al*, "Application and network support for InfoPad", *IEEE Personal Communications*, 3, 2, March 96.

[10] B. N. Noble and M. Satyanarayanan, "A research status report on adaptation for mobile data access", *SIGMOD Record*, 24, 4, Dec. 1995.

[11] 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.

[12] M. Satyanarayanan, "Fundamental challenges in mobile computing", *ACM Symposium on Distributed Computing*, Philadelphia, Pa., May 1996.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.