# Triggering information by context

P. J. Brown

Computing Laboratory,

The University of Kent at Canterbury

Canterbury

Kent CT2 7NF

England

pjb@ukc.ac.uk

**ABSTRACT**

With the increased availability of personal computers with attached sensors to capture their environment, there is a big opportunity for *context-aware* applications; these automatically provide information and/or take actions according to the user's present context, as detected by sensors. When well designed, these applications provide an opportunity to tailor the provision of information closely to the user's current needs. A sub-set of context-aware applications are *discrete* applications, where discrete pieces of information are attached to individual contexts, to be triggered when the user enters those contexts. The advantage of discrete applications is that authoring them can be solely a creative process rather than a programming process: it can be a task akin to creating simple web pages.

This paper looks at a general system that can be used in any discrete context-aware application. It propounds a general triggering rule, and investigates how this rule applies in practical applications.

KEY WORDS: context-aware; triggering; information filtering; PDA; sensor

As personal computers become more aware of their environment, a whole new range of applications, called *context-aware* applications [Schilit *et al*, 1994], become attractive. The basic aim of this paper is to look at:

- how context-aware information can be represented in a way that makes authorship of applications simple.
- what the rules for triggering information by context should be.

We expand on these aims later, but first we will describe context-aware applications in general, and show the wide range they cover.

Context-aware applications typically focus on a mobile user who is carrying a Personal Data Assistant (PDA) with environmental sensors. These sensors could cover:

- location: as detected by GPS, active badges [Want *et al*, 1992], PARCTabs [Adams, 1993, Want *et al*, 1995], mobile phones [Duffet-Smith, 1996] or ultrasonic techniques [Ward *et al*, 1997].
- companions or objects nearby, as detected by active badges or tags.
- orientation, as detected by an electronic compass.
- time of day and time of year, as are ubiquitously available on computers.
- data captured by specialised sensors for particular applications, such as temperature, radiation level, or even share prices (where the 'sensor' is an information feed from a stock exchange).
- data synthesised by 'super-sensors' which correlate information from lower level sensors in order to deduce some higher level state; for instance a super-sensor might detect whether the occupant of an office is busy by looking at sensors that record what equipment is being used, who else is in the office, etc.

The values of these sensors together constitute the *present context* of the user. Many context-aware applications work like this: there is a *repository* of information, consisting of individual *notes*, each of which is a piece of information attached to a particular context. (We use the term 'note' as a parallel to a Post-it note, which is a piece of information physically attached to a context.). A set of related notes, collected together as a

unit, is called a context-aware *document*. When the user's present context matches the context on a note the note is *triggered*. Triggering can cause various actions, depending on the application, e.g.:

- presenting information to the user.
- running a program: here the information attached to a context is a program or a script.
- configuring the screen of the user's PDA: here the information attached to a context is some form of layout specification.

In most context-aware applications triggering is automatic, i.e. there is server push rather than client (user) pull: the idea is to present the user with relevant information that the user did not otherwise know about. Nevertheless the user needs some control, e.g. to tame or to silence over-enthusiastic triggering, and in some applications triggering is directly under the user's control.

Some examples of context-aware applications are:

- tourism [Long *et al*, 1996]: presenting the user with information about sights she is passing. There could be many context-aware documents that the user could choose from, e.g. covering various cities, or various buildings within a site.
- equipment maintenance: logically similar to tourism, but oriented to field engineers and the equipment they need to be aware of.
- ecological fieldwork: capturing and presenting information.
- 'attaching' information to tagged farm animals or tagged equipment: here the information might really reside in the PDA, but is triggered when the object of interest is nearby, thus giving an illusion that the information is somehow attached to an object.
- 'real' timetables for public transport: here the application knows the position of the user and of the public transport vehicles, and can guide the user to the nearest stop, to arrive shortly before the public transport vehicle does.
- personally tailored control panels, which pop up on a PDA as the user passes office equipment such as a photocopier, and, using an infra-red link say, allow the user to control the equipment.
- teleporting [Bennett *et al*, 1994]: automatic bringing up of the user's desktop on nearby computers as the user moves around.
- applications in modelling and virtual reality. The 'sensors' are easy to provide in such environments, because the whole world is created by the computer. A specially promising application is information that relates both to the modelled world and to the corresponding real world. For example, information attached to the location of a stairway on an oil-rig can be triggered via GPS or DGPS on the rig itself, and by the user's simulated position as he moves around the corresponding model.

All these applications can be based on discrete pieces of information. There are also more advanced applications where the information shown to the user changes continually as their context changes. A simple example would be one that continually showed the user's distance from a given goal. Such *continuous* applications cannot be modelled in terms of retrieving discrete pieces of information; instead their creation requires serious programming effort. We concentrate just on discrete applications in this paper since they are the easier to analyse: fortunately this still covers a vast field, though there is also a vast number of continuous applications, which are not covered here.

The advantage of discrete applications is that they can all be encompassed by a single general mechanism: the author creates notes in the appropriate format, and a general-purpose triggering engine triggers these when they match the user's present context. Thus authorship becomes an exercise in creative skills rather than programming, just as, for example, authoring web pages.

In detail the overall model is that the author creates notes and aggregates these into documents. One or more documents are loaded into the triggering engine. The triggering engine continually checks all its notes against the present context, and triggers those that match. The triggering engine sends the triggered notes to a client application, which interfaces with the user. The only programming that is needed is to create the application, with its user interface, and to set up a mechanism so that sensors can continually update the present context; once this infrastructure is in place, any author can create documents without programming effort. Most applications will be *special-purpose* in that they will have a user interface designed for a particular purpose, e.g. tourism or ecological fieldwork. It is possible, however, given a general triggering mechanism, to have a corresponding *general-purpose* application that presents all kinds of triggered information to the user. There is the usual trade-off between general and special purpose: on the one hand there are the advantages of generality and on the other of tailoring the user interface to a particular purpose. As an example of the latter, Figure 1 [Pascoe *et al*, 1997] shows the interface for one application, and Figures 2 and 3 [Ryan *et al*, 1998] for another: a key requirement of the first application is that the screen should have an extremely simple interface, since the user's attention would be on observing moving animals, whereas in the second that user is working with static material, and can afford to spend time concentrating on the screen.

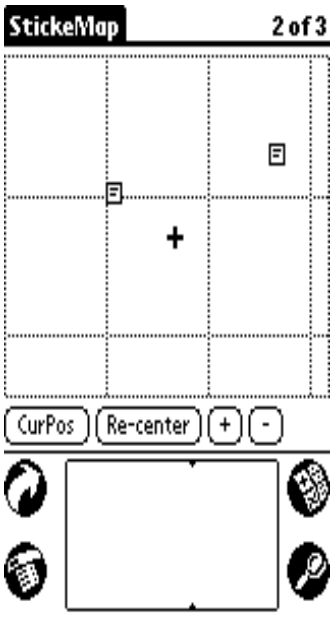


*Figure 1: the user interface for an ecology application: the two small square icons represent points for which information is available; the cross shows where the user is*

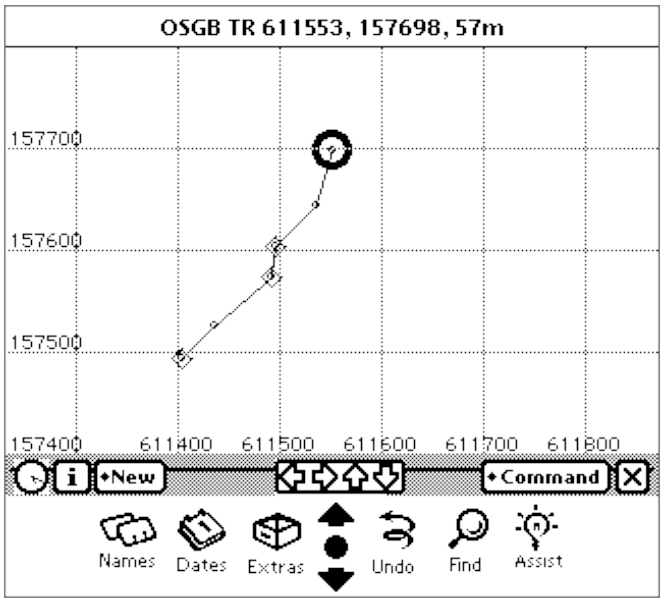


*Figure 2: the user interface for an archaeological application; here there is a more elaborate interface, where the user's track is shown.*

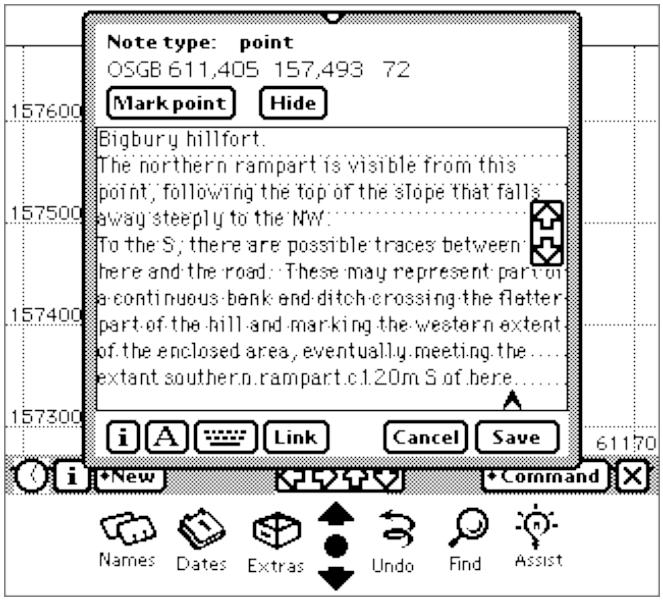


*Figure 3: shows the change in Figure 2 when the map is partially covered by a triggered note*

## *Pretended fields and values*

Almost all context-aware applications are enhanced by augmenting the 'real' world, as detected by sensors, by a 'pretended' world, controlled by the user and/or application. There are several examples of this.

Firstly consider a location-aware application used as a tourist guide. Clearly it is a limitation if the only way a tourist can see information about a place is to actually go there (not forgetting to take a GPS receiver attached to their PDA). It is much better if the user can, as an alternative, pretend to be at the place, and thus trigger the information: they can then judge whether the place is worth visiting. In the user interface, pretence can be implemented by allowing the user to point at a map, thus pretending they are at the location pointed at, and temporarily overriding the real setting for their location as given by a sensor.

Secondly, fields of the present context can relate entirely to imaginary concepts, with no real sensor at all. (We later explain in detail what fields are.) Continuing our example of a tourism application, it might have an optional extra field with the name 'companion'. The author provides some notes of special interest to architects, and these all have the imaginary companion 'Architect' as part of their context; a similar approach is used for other special companions, e.g. 'Historian'. The user can then pretend they are with one or more of these, by adding to their present context the information that they are with, for example, an Architect: the notes of special interest to architects are then triggered, provided, of course, that their other context fields are matched too, e.g. their location. Companions can also be used to implement tours: for example every note on a certain tour might have One-hour-tour-guide as an imaginary companion, and the user with this companion in their present context will see the notes for the tour – these will doubtless include notes that tell the user how to get from one sight on the tour to the next. This use of imaginary companions follows on from a technique described by Oren *et al* [1990], which was concerned with the use of imaginary companions to guide users though a large body of information concerned with American history.

Thirdly, values of fields can go beyond what is possible in reality. We discuss this later.

## *Retrieving information*

Our aim in this paper is to look at what triggering behaviour is needed by authors and users of discrete context-aware applications, rather than to focus on how triggering is implemented, e.g. whether there is an underlying information retrieval engine or database management system. We now consider how a general-purpose triggering mechanism can work: its basic task is to match the user's present context with the contexts and associated information supplied by the author. We would like to have a matching algorithm which gives the user just the information he wants, with no surprises and no overloading, and makes the authorship of context-aware information straightforward. In addition there are some more specific needs:

1. users should not need to understand the mechanism by which triggering is done: to them triggering is something that just happens. For simple applications, even authors should need only a minimal understanding. (Often, of course, the author and the user is the same person.)
2. both users and authors should be able to exchange and publish either single notes or whole documents.
3. if the user adds a new sensor, a temperature sensor say, to augment her present context, then information that is *not* concerned with temperature should be triggered exactly as before. Information that *is* concerned with temperature, however, should be triggered more

precisely than before, e.g. if the temperature was low, she would *not* get information that the author had marked as relevant at high temperatures. Given this need, we have a sequence of further needs, each following from its predecessor.

4. if the author discovers that users have a new form of sensor – again we will assume a temperature sensor – that was not allowed for in the original context-aware document, the amount of change required to cater for this should be minimal. In particular no existing note in the document should *require* changing, though the author might *choose* to refine some of the notes by adding a temperature field that gives more focussed triggering. For example information about the joys of boating on the river might be suppressed if the temperature was currently below freezing. The author can also choose to add extra notes to cater for the new sensor; this should not, however, adversely affect users who did not have the sensor.
5. context-aware documents should not be tied to users who have a particular set of sensors. Instead it should be straightforward to write a document that is suitable for a diverse set of users, having a diverse set of sensors. There may, however, be minimal requirements: many documents would be virtually useless in the field without some sort of location sensing.
6. if one of the user's sensors fails, the triggering behaviour should continue in as reasonable a way as possible, and the author should be able to give special information to be triggered in such cases.

## *The matching algorithm*

We now consider the matching algorithm to meet these needs. The design of the algorithm is the essence of triggering. The choice of algorithm is not obvious, because context-aware retrieval is different from ordinary retrieval.

We will start by defining an overall model, which aims to be general enough to cover most applications, and most sorts of data – either existing data such as previous fieldwork readings or data specially created for a particular context-aware application. It is the model now used by *stick-e notes* [Brown *et al*, 1997], which is one technology for the creation of context-aware applications.

In addition to the matching algorithm, a key part of the model is the way data is represented. The method used to represent data is a simple one – even a simplistic one: each note just consists of a set of *field*s, each of which is a name/value pair. The meanings of the fields, e.g. whether they are context or information, is not fixed, but can be determined by the application. In the examples here we will show names of fields as SGML tags. A sample note is:

*Note A:*

```
<location> X

<body> This is the Westgate.
```

where X is a certain latitude/longitude, or other representation of location. We have called this 'note A', so that we can refer to it later. The repository contains all the notes that are currently active, i.e. poised for triggering. In some applications the repository will be quite static, whereas in others, e.g. traffic information, it will be changing all the time.

The initial implementation of stick-e notes made the mistake of fixing, at time of authorship, how notes were to be retrieved, e.g. that <location> fields were part of the context to be matched, and <body> fields represented triggered information. In short, the author was required to wire in a lot of semantic information. It soon became clear, initially as a result of fieldwork applications created by Pascoe [Pascoe *et al*, 1997], that users want to trigger context-aware material using all sorts of criteria, just as they use all sorts of criteria for normal information retrieval. For example, they may want any note whose <body> field contains the word 'snake' to be triggered automatically when its location is matched, irrespective of any other fields – thus the <body> field is used as a criteria for triggering (and indeed may also be used as the information triggered). Stick-e notes now use the simpler model of a set of name/value pairs; this has two advantages:

- users/applications, not authors, control how information is triggered. Often this is done in ways the author never dreamed of. The author can, however, make suggestions on how their work may be used: as we shall see, this might involve suggested default values for fields within the present context.

- since no great constraints are placed on the form of notes, existing data, not produced with context-aware triggering in mind, can still be used for context-aware triggering. On the other side of the same coin, data created for context-aware retrieval can be kept, possibly after a bit of conversion, in existing data repositories.

Within this model the present context is also a note, i.e. a set of name/value pairs. Some or all of these fields will represent current values set by sensors (or pretended sensors). Thinking of the present context as a note brings some benefits of symmetry: for example if we save the present context, every ten minutes say, then the set of saved notes can itself act as a repository for later retrieval. This can be valuable for field applications.

We assume in this paper that each field within a note has a unique name. In fact this need not be so, but the detail is a distraction.

We are now ready for the first example: assume that the user's PDA has a single sensor, a location sensor, and that the present context is

*Present context P:*

```
<location> Y
```

The natural rule is that note A above should trigger if location X matches location Y. The rules for determining whether two values match will be set by the application. Typically two possible approaches are used for matching locations:

1. both locations are points, and the two are considered to match if they are within a certain distance of each other.
2. at least one of the locations is a region, e.g. a circle or rectangle, rather than a point, and two locations match if they overlap.

In most of the applications we have worked with, fields on notes, if numeric, specify ranges of values rather than single values, as in approach (2) above. Thus a criteria for triggering might be a temperature in the range 30..40, rather than, say, exactly the temperature 35. (In some cases matching of field values may involve boolean operators such as AND and OR, string-matching operators, etc., but we will not discuss such detail here.)

When a note has been matched, triggering often consists of bringing to the user's attention those fields that did not take part in the matching process. In the above example this is just the <body> field of note A.

As a second example, assume that the user now has a (Centigrade) temperature sensor fitted to her PDA. Thus the present context might be

*Present context Q:*

```
<location> Y

<temperature> 25
```

A natural rule, in line with requirement (3) for retrieving information, is that note A should still trigger if its location is matched, irrespective of the current temperature. More generally, a field of the present context that is not mentioned in a corresponding note is considered as irrelevant to the matching.

We now consider a second note, note B, which is general information about mild temperatures. This note consists of

*Note B:*

```
<temperature> 10..30

<body> No extra maintenance schedule is needed.
```

Following the same rules, this note will match present context Q above, since the values of the temperature field match. In this match it is the location field of the present context that is irrelevant rather than the temperature field, since note B does not have a location field.

Now consider a third note, note C, which is concerned with maintenance, at certain temperatures, of a particular machine, located at X. Assume this consists of

*Note C:*

```
<location> X

<temperature> 0..10

<body> Add antifreeze, by pouring it

in the blue chamber on top of this machine.
```

Given that both note C and the present context contain a location field and a temperature field, a natural rule is that both these fields should match if note C is to be triggered. If, incidentally, the user did not have a temperature sensor, note C would be triggered if the user's location matched: we have suggested there be a default rule that the information which the user sees in this case should be the fields that were not involved in the match. Thus the user might see on the screen something like

```
            0..10

      Add antifreeze …
```

This ends the sequence of examples: their purpose has been to suggest what the user would like as a default triggering behaviour.

## A matching rule

We will now try to generalise the behaviour illustrated by the above examples and formulate a rule for matching the present context with each note. The rule first involves defining the names of the fields of the present context that will form part of the matching process. We call these fields the *retrieval-awareness*. Normally the retrieval-awareness fields will be the fields corresponding to sensors, real or pretended. As an example, we have assumed that, in the present context Q above, the retrieval-awareness fields are location and temperature, whereas in present context P it is just location. The retrieval-awareness fields will be set by the application, doubtless with some interaction from the user. However, the user should not need to be aware of the concept of retrieval-awareness, but should work in more problem-oriented terms. The constituents of the retrieval-awareness may change dynamically, e.g. when the user says 'from now on, take the time field into account'. Given the concept of retrieval-awareness, the rule for matching a note N with the present context is as follows:

- *for each field F within note N that has the same name as a retrieval-awareness field, the value of F must match the value of the corresponding retrieval-awareness field.*

## Relation with general information filtering and retrieval

It is instructive to relate context-aware retrieval to the general fields of information filtering and retrieval. DeClaris [DeClaris *et al*, 1994], following on from Belkin and Croft [1992], categorises these general fields as follows: "Retrieval and filtering are two extremes of a continuum. In information retrieval user's queries may vary significantly during a single session, while the collection of information to be searched is relatively static. In information filtering, a user's interests are relatively static, but those interests are matched against a dynamic information stream, such as newly published medical journals or conference proceedings". Context-aware retrieval indeed lies in this (multi-dimensional) continuum. It is similar to information filtering in that the present context, and the changing values of the fields within this, represent the dynamic information stream, while the notes are the equivalent of a set of different user interests (actually they are the interests, as the author sees them, of the *same* user in different contexts); however what is retrieved, i.e. shown to the user, is the note, not the retrieval-awareness – the opposite way round to information-filtering.

Fortunately context-aware retrieval is in many ways a much easier case than general retrieval, especially when our simple models for information representation and matching are used. In particular:

- the information to be retrieved is in a uniform form (name/value pairs).
- matching of values is normally simple: testing if two locations match is much easier than, say, determining if a medical article relates to disease rates in nomadic communities.
- the statement of the user's interest, i.e. the retrieval-awareness, can be created simply and mechanically from the sensors, real or pretended.

Because the problem is so much simpler, it is possible to find simple solutions, such as our suggested matching algorithm, to what is in the general case an extremely challenging problem.

## Combining with normal retrieval

A general lesson in the retrieval field is that no one approach covers all needs, and a combination of approaches is often needed. This applies strongly to context-aware retrieval, and a good partner is often conventional information retrieval – the form of retrieval that is called IR (or just 'retrieval' in DeClaris' definition). Thus as well as an retrieval-awareness, which controls context-aware retrieval, the user might like to have, within the present context, a further *retrieval-selector*, which uses ordinary retrieval techniques to select from the notes that match the retrieval-awareness. For example the retrieval-selector might say the <body> field must contain a certain word and/or that there must be a location field that matches a certain area ("I only want to see triggered notes that lie within [the limits of the city wall]"). These two forms of retrieval working together can be a powerful aid to meeting the user's needs. The order in which the two forms of retrieval are applied is an implementation matter, but for the purpose of this discussion we will assume that the retrieval-selector applies to the notes triggered through context-aware retrieval; thus a retrieval-selector may cut out some or all of the triggered notes.

## Triggering once

Another way that context-aware retrieval is different from ordinary retrieval is that notes need triggering just once. For example assume that the triggering engine is being regularly asked to supply notes to the application. There is one note covering the whole area of a certain city; this is triggered when the user enters the city, and this is passed to the application, which will doubtless make the user aware of the existence of the triggered note. It is not desirable for this note to be continually re-triggered all the time the user is in the city; thus the job of the triggering engine is just to trigger new notes which have not been triggered before. There are some sub-issues here, when the user leaves the city, concerning possible *de-triggering* (i.e. telling the application that a triggered note is no longer in context) and then re-triggering if the user later re-enters the city. These issues need not concern us here; in the design of most applications, de-triggering is not a requirement, though re-triggering on leaving and then re-entering a context might be.

## Enhanced retrieval using pretended values

We have highlighted the use of pretended values and fields to give the user more control over the retrieval of context-aware notes. In order to enhance this, pretended values can be carried one stage further by extending reality. A simple example of this is that a user can pretend they are in a whole area, a nature reserve perhaps, thus triggering all the notes for that area and allowing the user to preview the complete scene. This extension of reality contradicts the saying 'you cannot be in two places at one'. Going a stage further, it is possible to have *metavalues* like `none`, `any` (meaning all values) and `not-working` (an error condition when there is a problem with a sensor). These metavalues can be used to set the present context, and can also be used on notes, e.g. a note:

```
<location> not-working

<body> Your location is currently unknown,

but this application

also supplies a lot of information that

is not dependent on location.
```

(Internally, these metavalues can be implemented as 'impossible' values, or all-encompassing ranges; this may be necessary if an existing retrieval engine is being used to implement context-aware triggering.)

As we see in the next section, the metavalue `none` can have an important role in limiting the amount of context-ware information that is triggered.

## *Drowning and dams*

The matching rule has an element of 'when in doubt trigger'. For example if the present context has no temperature field in its retrieval-awareness, as in present context P, then note C, which gives information on what to do at a certain temperature and at a certain location, is triggered when the user visits the location. The user then makes their own judgement of whether the information is relevant – for temperature the user can probably make an informed guess as to what the current value is. In many applications this triggering of potentially irrelevant information is an important safeguard, but in others the user may regard the flood of triggerings to be as welcome as SPAM e-mail.

Overall therefore the user needs to have a mechanism to prevent floods of triggerings. The mechanism is, in fact, quite simple: use metavalues. If, for example, the application places

```
<share> none
```

as an retrieval-awareness field in the present context, then no note containing a share field will be triggered (because the metavalue `none`, by definition, is a mismatch with any value). As a second example, if the user/application places

```
<companion> none
```

as an retrieval-awareness field of their present context, then specialised notes with a pretended companion such as Architect or Historian will not be triggered.

Overall, therefore, the use of `none` fields represents an effective dam to prevent a flood of information. Indeed the sea of triggered information might be cut down to a small puddle, containing just the information that is absolutely relevant. As we have emphasised, our philosophy is that retrieval is under the control of the end-user rather than the author, but we bend this a little by allowing authors to make suggestions of retrieval-awareness fields and their default values, to be used if the field does not already exist in the present context. Thus our set of notes for tourists may suggest

```
<companion> none
```

be a default value in the present context, and that this be a retrieval-awareness field. The suggestion can be applied, or rejected, by the application/user when the set of notes is loaded into the repository.

## *The advantage of using two types of retrieval*

Consider a note of special interest only to architecture buffs:

*Note D:*

```
<location> X

<companion> Architect
```

`<body> …`

If the user wants specialised notes such as this to be triggered, and in addition to have other notes triggered normally, they can place

`<companion> Architect`

as an retrieval-awareness field in their present context. Thus they are using the normal context-aware retrieval mechanisms. If, on the other hand, the user *only* wants to see notes of special interest to architects, then they use ordinary information retrieval methods to select what they want, i.e. they specify a filter that only lets through notes that have an architect as a companion. Specifically they place

`<companion> Architect`

as a retrieval-selector field of the present context, and this is applied in addition to context-aware retrieval (which, if location was part of the retrieval-awareness, would trigger the notes whose location field matched). Thus, by exploiting the two types of retrieval, the user has flexibility in what they trigger. The user need not, of course, be aware of all these machinations: they can just have buttons like 'Architect only' and 'Architect as well', which, when selected, make the necessary settings of the retrieval-awareness and retrieval-selector fields of the present context.

(There is, however, one detail that requires resolving: there is an ambiguity as to whether a note such as note D is (a) only of interest to architects, or (b) both of general interest and of interest of architects. There must be something to distinguish notes of type (a) from those of type (b). In our model the difference manifests itself when the user has, as a retrieval-awareness field, a companion other than an architect. In this situation the user should still trigger notes of general interest (type (b) above). However the matching rule says that note D should not be triggered when the companion in the present context does not match Architect. There are ways round this using attributes, etc., but it is an unsatisfactory detail.)

## *Implementation*

Several applications have been created using these principles: the most developed one in terms of triggering is a tourist application created by Xian Chen. In addition, Jason Pascoe and Nick Ryan have created applications, in ecology and archaeology respectively, where the accent is on capturing information, which might then be used for later triggering. These implementations have not been built round existing database or retrieval systems, partly because a PDA represents a constrained environment and partly because the matching and triggering processes are straightforward to code in a general and portable way without relying on existing software. Nevertheless several of our applications need to relate to data stored in databases, and we interface with them via uploading and downloading: e.g. store notes in the database; download them into the PDA at the beginning of a trip; use them for triggering during the trip; at the end of the day, if the notes have been changed, upload them back into the database.

In addition to the applications that have been used in practice, we have created prototypes to show the generality of the model over a wide range of areas, ranging from share price triggering to attaching information to tagged animals.

## *Conclusions*

To summarise, the purpose of this paper has been to present a general model for discrete context-aware applications. The model covers the way information is represented and how it is triggered. The model allows context-aware retrieval to be combined with ordinary retrieval, thus giving the user great flexibility in controlling what they want to be triggered. There are, we believe, numerous applications waiting to be exploited, particularly those that bring together many contextual fields in order that triggered information is tied ever closer to the users' contexts and thus to what they are really likely to want.

## *Acknowledgements*

My colleagues John Bovey and Xian Chen, working under an EPSRC grant, have contributed greatly to our research programme in context-aware applications. In addition, Nick Ryan, David Morse and Jason Pascoe, working under a JTAP grant, have been concerned with applying these ideas to applications in fieldwork, and have come up with many insights. Outside the University of Kent, I would particularly like to note the contributions of staff from Xerox Research Centre Europe at Cambridge, and the anonymous referees for their comments.

## *References*

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

Belkin N J and Croft W B. (1992) Information filtering and information retrieval: two sides of the same coin, *Comm. ACM* 35(12), 29-38.

Bennett F, Richardson T, and Harter, A. (1994) Teleporting – making applications portable, *Proceedings of the Workshop on Mobile Computing*

*Systems and Applications*, Santa Cruz, Ca., IEEE Computer Society Press, Los Alamitos, Ca., 82-4.

Brown P J, Bovey J D and Chen X. (1997) Context-aware applications: from the laboratory to the marketplace, *IEEE Personal Communications* 4(5), 58-64.

DeClaris N, Harman D, Faloutsos C, Dumais S and Oard D. (1994) Information filtering and retrieval: overview, issues and directions, *Proceedings of the 16th Annual International Conference of the IEEE Engineering in Medicine and Biological Society*.

Duffet-Smith P. (1996) High precision CURSOR and digital CURSOR: the real alternatives to GPS. Proceedings of EURONAV 96 Conference on Vehicle Navigation and Control, Royal Institute of Navigation.

Long S, Aust D, Abowd G D, and Atkeson, C. (1996) Cyberguide: prototyping context-aware mobile applications, *CHI'96* short paper, 293-4.

Oren T, Solomon G, Kreitman K, and Don A. (1990) Guides: characterizing the interface, in Laurel B (Ed.), *The art of human computer interface design*, Addison-Wesley, Reading, Mass., 367-81.

Pascoe J, Morse D R and Ryan N S (1997) Developing personal technology for the field, *Computing Laboratory, University of Kent at Canterbury*, submitted for publication.

Ryan N S, Pascoe J and Morse D R. (1998) Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant, in V. Gaffney, M. van Leusen and S. Exxon (eds.) *Computer Applications in Archaeology 1997*, published March 1998.

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

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

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

Ward A, Jones A and Hopper A. (1997) A new location technique for the active office, *IEEE Personal Communications* 4(5), 43-7.