Technical Report GIT-GVU-98-33

The Context Toolkit: Aiding the Development of Context-Enabled Applications

Daniel Salber, Anind K. Dey, Gregory D. Abowd
GVU Center, College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280
+1 404 894 7512
{salber, anind, abowd}@cc.gatech.edu

ABSTRACT

Context-enabled applications are just emerging and promise richer interaction by taking environmental context into account. However, they are difficult to build due to their distributed nature and the use of unconventional sensors. The concepts of toolkits and widget libraries in graphical user interfaces has been tremendously successful, allowing programmers to leverage off existing building blocks to build interactive systems more easily. We introduce the concept of context widgets that mediate between the environment and the application in the same way graphical widgets mediate between the user and the application. We illustrate the concept of context widgets with the beginnings of a widget library we have developed for sensing presence, identity and activity for people and things. We assess the success of our approach with two example context-enabled applications we have built and an existing application to which we have added context-sensing capabilities.

Keywords

Context-enabled or context-aware computing, ubiquitous computing, toolkits, widgets, applications development

INTRODUCTION

Over the last decade, several researchers have built applications that take advantage of environmental information, also called context, to enhance the interaction with the user. The construction of these context-enabled applications is cumbersome, and currently no tools are available to facilitate the development of this class of applications. This paper presents a toolkit for developing reusable solutions for handling this context information in interactive applications.

We first define the notion of context, and through a brief review of the literature identify the key challenges of developing applications that sense context, followed by an overview of the paper.

Paper submitted to CHI'99 for review.

What Is Context?

Environmental information or context covers information that is part of an application's operating environment and that can be sensed by the application. This typically includes the location, identity, activity and state of people, groups and objects. Places such as buildings and rooms can be fitted with sensors that provide measurements of physical variables such as temperature or lighting. Finally, an application may sense its software and hardware environment to detect, for example, the capabilities of nearby resources.

Sensing context information makes several kinds of context-enabled applications possible: Applications may display context information, capture it for later access and provide context-based retrieval of stored information. Of major interest are context-aware applications, which sense context information and modify their behavior accordingly without explicit user intervention.

What Are Context-Enabled Applications Used For?

Usage scenarios of typical context-enabled applications found in the literature have led us to identify recurrent challenges, which we further detail below.

Mobile tour guides are designed to familiarize a visitor with a new area. They sense the user's location and provide information relevant to both the user and the location she's at [1, 3, 7, 11]. Likewise, office awareness systems sense users' locations, but are also interested in their activities to e.g., help people locate each other, maintain awareness or forward phone calls [13, 16, 17]. In ubiquitous computing systems, devices sense and take advantage of nearby resources: a handheld computer located next to an electronic whiteboard may make use of the larger display surface or allow the user to interact with other nearby handheld users [15, 17]. Finally, context-based retrieval applications gather and store context information and allow later information retrieval based on context information. For instance the user can ask a note-taking application to pull up the notes taken at a previous meeting with the group she's meeting with currently [10, 14].

What Makes Context-Enabled Applications Difficult?

The above usage scenarios raise the following difficulties, which are common to most applications that use context information. These difficulties stem from the very nature of context information:

1) It is acquired from unconventional sensors. Mobile devices for instance may acquire location information from outdoor GPS receivers or indoor positioning systems. Tracking the location of people or detecting their presence may require Active Badge devices, floor-embedded presence sensors or video image processing.
2) It must be abstracted to make sense for the application. GPS receivers for instance provide geographical coordinates. But tour guide applications would make better use of higher-level information such as street or building names. Similarly, Active Badges provide IDs, which must be abstracted into user names and locations.
3) It may be acquired from multiple distributed and heterogeneous sources. Tracking the location of users in an office requires gathering information from multiple sensors throughout the office. Detecting the presence of people in a room may rely on the combination of video image processing, audio signal processing, floor-embedded pressure sensors, etc.
4) It is dynamic. Changes in the environment must be detected in real time and applications must adapt to constant changes. For instance, when a user equipped with a handheld moves away from the electronic whiteboard, the user loses the benefit of the wide display surface and the application must modify its behavior accordingly.

Although these problems are recurrent we lack conceptual models and tools to describe solutions to these problems.

Overview of paper

This paper presents a context toolkit aimed at developing reusable solutions to address these problems and thus make it easier to build context-enabled applications. The inspiration for this context toolkit is the success of toolkits in graphical user interface (GUI) development. The context toolkit builds upon the widget concept from GUI toolkits. In the same way GUI toolkits insulate the application from interaction details handled by widgets, the context toolkit insulates the application from context sensing mechanics through widgets. We first introduce the concepts underlying the context toolkit and describe applications it allowed us to build. We discuss details of the toolkit implementation that address specific problems of distribution, heterogeneity and dynamism that are not addressed by GUI toolkits. We conclude with future plans for the evolution of the context toolkit.

DESIGNING A CONTEXT TOOLKIT

The context toolkit we have developed relies on the concept of context widgets. Just as GUI widgets mediate between the application and the user, context widgets mediate between the application and its operating environment. We analyze the benefits of GUI widgets, introduce the concept of the context widget, detail its benefits, and explain how context-enabled applications are built using these widgets.

Learning From Graphical User Interface Widgets

It is now taken for granted that GUI application designers and programmers can reuse existing interaction solutions embodied in GUI toolkits and widget libraries. GUI widgets (sometimes called interactors) span a large range of interaction solutions: selecting a file; triggering an action; choosing options; or even direct manipulation of graphical objects [12].

GUI toolkits have three main benefits:

- They *hide specifics* of physical interaction devices from the applications programmer so that those devices can change with minimal impact on applications. Whether the user points and clicks with a mouse or fingers and taps on a touchpad doesn't require any change to the application.
- They *manage the details* of the interaction to provide applications with relevant results of user actions. Widget-specific dialogue is handled by the widget itself, and the application often only needs to implement a single callback to be notified of the result of an interaction sequence.
- They *provide reusable building blocks* of presentation to be defined once and reused, combined, and/or tailored for use in many applications. Widgets provide encapsulation of appearance and behavior. The programmer doesn't need to know the inner workings of a widget to use it.

Although toolkits and widgets are known to have limitations such as being too low-level, they provide stepping stones for designing and building user interfaces and developing tools such as User Interface Management Systems (UIMS). With context widgets, we aim at providing similar stepping stones for designing and building context-enabled applications.

What Is A Context Widget?

A context widget is a software component that provides applications with access to context information from their operating environment. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns.

Context widgets provide the following benefits:

- They *hide the complexity* of the actual sensors used from the application. Whether the presence of people is sensed using Active Badges, floor sensors, video image processing or a combination of these should not impact the application.
- They *abstract context information* to suit the expected needs of applications. A widget that tracks the location of a user notifies the application only when the user moves from one room to another, or from one street corner to another, and doesn't report less significant moves to the application. Widgets provide abstracted

information that we expect applications to need the most frequently.

- They *provide reusable and customizable building blocks* of context sensing. A widget that tracks the location of a user can be used by a variety of applications, from tour guides to office awareness systems. Furthermore, context widgets can be tailored and combined in ways similar to GUI widgets. For example, a *Presence* widget senses the presence of people in a room. A *Meeting* widget may rely on a *Presence* widget and assume a meeting is beginning when two or more people are present.

These benefits address issues 1 and 2 listed in the introduction. From the application's perspective, context widgets encapsulate context information and provide ways to access it in a way very similar to a GUI toolkit. However, due to the characteristics of context and notably issues 3 and 4 mentioned in the introduction, the context toolkit has some unique features. We briefly describe the similarities with GUI toolkits and point out some major differences.

How Applications Use Context Widgets

Context widgets have a state and a behavior. The widget state is a set of attributes that can be queried by applications. For example, an *IdentityPresence* widget has attributes for its location, the last time a presence was detected, and the identity of the last user detected. Applications can also register to be notified of context changes detected by the widget. The widget triggers callbacks to the application when changes in the environment are detected. The *IdentityPresence* widget for instance, provides callbacks to notify the application when a new person arrives, or when a person leaves.

So far, context widgets are very similar to GUI widgets, however there are important differences:

- Context widgets live in a distributed architecture because context may need to be acquired from multiple distributed sources. Widgets are made up of three distributed components: generators that acquire context information, interpreters that abstract it and servers that aggregate information. Applications, widgets and the components of a widget themselves may be distributed. This feature of the toolkit addresses issue 3 listed in the introduction
- Context widgets monitor environmental information that may be needed at any time by an application. Thus a context widget is active all the time, and its activation is not, as with GUI widgets, driven by applications. This feature, along with other characteristics of the toolkit described in the implementation section, addresses issue 4.

The context toolkit aims at enabling easier development of context-enabled applications. To assess our objective, we have built context widgets we expect applications to need frequently and have developed applications based on these widgets.

BUILDING CONTEXT WIDGETS

In this section, we describe two of the context widgets that we have built. These context widgets aim at surveying an indoor environment. We show examples of their use in the applications described in the next section. The first widget, *IdentityPresence*, is attached to a specified location and senses the surrounding environment for the presence of people. The second widget, *Activity*, continuously monitors the surrounding environment for significant changes in activity level.

The *IdentityPresence* widget

The *IdentityPresence* widget is placed in a pre-specified location and it reports the arrival and departure of people at that location. The identities of the people arriving and departing as well as the times at which the events occurred are also made available to applications. The information this widget provides is useful for any location-aware application (e.g. tour guide, tracking people).

The *IdentityPresence* widget provides applications with the attributes and callbacks listed in Table 1.

Widget Class	IdentityPresence
Attributes	
Location	*Location the widget is monitoring*
Identity	*ID of the last user sensed*
Timestamp	*Time of the last arrival*
Callbacks	
PersonArrives (location, identity, timestamp)	*Triggered when a user arrives*
PersonLeaves (location, identity, timestamp)	*Triggered when a user leaves*

Table 1. Definition of the *IdentityPresence* widget.

All widgets require entities to generate their information from sensors. Appropriately, these entities are called *generators*. A generator encapsulates a single sensor and the software that acquires raw information from the sensor.

The *IdentityPresence* widget could be implemented using any number of generators, including voice recognition, Active Badges, video/image recognition, keyboard and login information, or even a combination of these. The generator that is chosen affects neither the definition of the widget nor any application that uses the widget. The attributes and callbacks provided by the widget are independent from the actual implementation, thus sheltering the application from the specifics of the sensors used. Our current implementation of the *IdentityPresence* widget uses Dallas Semiconductor's iButtons [5], passive tags with unique identifiers and storage and computing capabilities.

The *Activity* Widget

The *Activity* widget senses the current activity level at a location such as a room. It may be used to sense the presence of people if they are active in a room. While it can not provide reliable presence information by itself, it can provide additional environmental information, e.g., to sense that a group of people are leaving the room. The widget is instantiated at a pre-specified location. Applications that use the *Activity* widget specify parameters for receiving callbacks, as seen in the table below.

The attributes and callbacks supported by the *Activity* widget are listed in table 2.

Widget Class	Activity
Attributes	
Location	*Location the widget is monitoring*
Timestamp	*Time of the last change in activity level*
AverageLevel	*Activity level (none, some, a lot) averaged over a user-specified time interval*
Callbacks	
ActivityChange (location, AverageLevel, timestamp)	*Triggered when the activity level changes from one level to another*

Table 2. Attributes and callbacks of the *Activity* widget.

The *Activity* widget has been implemented with a microphone, but like the *IdentityPresence* widget, it could be implemented with any appropriate generator, such as an infrared sensor, video image analysis, or a combination of these.

Other Context Widgets

We have also constructed other widgets as part of the context toolkit. The *NamePresence* widget is similar to the *IdentityPresence* widget. Instead of providing an artificial user ID for a user whose presence has been detected, this widget provides the user's actual name. The *PhoneUse* widget provides information about whether a phone is being used and the length of use. The *MachineUse* widget provides information about when a user logs onto or off of a computer, his identity, and length of his computing session. The *GroupURLPresence* widget provides a URL relevant to the research group a user belongs to when her presence is detected. An application describing its use is given in the following section. The completeness and overall structure of the entire context widget library is an interesting open research issue.

BUILDING APPLICATIONS WITH THE CONTEXT TOOLKIT

In this section, we describe three applications we have built to assess the actual benefits of our context toolkit. To reiterate, the expected advantages of the toolkit are to hide complexity, provide appropriate interpretation of context information and ease overall construction through reusable widgets.

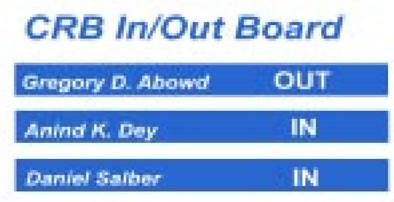
In/Out Board

Motivation

The first application we built was the electronic equivalent of a simple in/out board that is commonly found in offices. The board (see Figure 1) is used to indicate which members of the office are currently in the building and which are not. In both the academic and corporate world, we often find ourselves trying to determine whether someone is in the office, in order to deliver or receive necessary information.

Figure 1. Screenshot of the in/out board application

Context Information



The in/out board application is an example of a context-viewing application. It gathers information about the participants who enter and leave the building and displays the information to interested users. In particular, the context information is a participant's identity and the time at which they arrived or departed. This application is interested in events when people pass the single entry point into the building. Therefore, only a single instance of the *IdentityPresence* widget is required, and it is the one located at the entrance to the building. Through the use of this widget, the context-sensing infrastructure is successful in hiding the details of how the context is sensed from the application developer.

Future Extensions

With the use of additional *IdentityPresence* widgets located at strategic areas (e.g. offices, meeting spaces) within the building, the in/out board application can be extended to a PersonFinder application. This new application would display the location of users throughout the building on a building map.

Information Display

Motivation

For our second application, we built an information display, similar to those found in the literature [8, 18]. We aim to show that the context toolkit can be used to reimplement existing context-enabled applications. This application displays information relevant to the user's location and identity on a display adjacent to the user. It activates itself as someone approaches it, and the information it displays changes to match the user, her research group, and location.

Context Information

The context information used by the information display is the location of the display, the identity of the user, the research group the user belongs to and information that is interesting to that research group. A single *GroupURLPresence* widget is used to supply the information to this application. The widget installed nearest to the display is used. When a user's presence is detected by the widget, it makes a URL about the user's research group available to the interested application. The application shows the contents of the URL on the nearby display.

This application does not deal with the details of how the context information is sensed, meaning the widget is successful at hiding the complexity of the sensing infrastructure. As well, the widget provides the appropriate information and detail to the application. For example, this application could have been implemented with an *IdentityPresence* widget. This would have required the application to determine what research group the nearby user was in and find information relevant to that research group. However, using the *GroupURLPresence* widget alleviated the need to perform these extra steps.

Future Extensions

The information display application provides the beginnings of a simple tour guide application, where the user is mobile and displays are static. Essentially, a basic context-aware tour guide [1,7] displays information relevant to the location and the identity of the person viewing the information. The information display is a very simple example of a tour guide with only one location of interest, but with additional displays and information providing widgets, it could be extended to build a full tour guide application.

DUMMBO

Motivation

For our third application, we chose to augment an already existing system, the DUMMBO (Dynamic Ubiquitous Mobile Meeting Board) project at Georgia Tech [2]. DUMMBO is an instrumented digitizing whiteboard that supports the capture and access of informal and spontaneous meetings. Captured meetings consist of the ink written to and erased from the whiteboard as well as the recorded audio discussion. After the meeting, a participant can access the captured notes and audio by indicating the time and date of the meeting.

In the initial version of DUMMBO, recording of a meeting was initiated by writing or erasing activity on the physical whiteboard. In the revised, context-aware version of DUMMBO, we wanted to have recording triggered when a group of two or more people gathered around the whiteboard. We also wanted to use information about when people were present around the whiteboard and their identities to help in visualizing and accessing captured material.

Context Information

This application belongs to the context-aware class of applications. It uses context to modify its own behavior (e.g., automatically starting the recording when enough people are standing around the whiteboard). The context information used is the participants' identities, the time when they arrived at or left the mobile whiteboard, and the location of the mobile whiteboard. The application uses multiple *NamePresence* widgets, one for each location where DUMMBO could be moved to in our research lab, and one on DUMMBO itself to detect the presence of users. Once again, the application could have used *IdentityPresence* widgets but the *NamePresence* widgets provided the appropriate level of detail, requiring fewer steps on the part of the programmer and application.

Adding Context to DUMMBO

This application was augmented on both the capture side and the access side. On the capture side, information about how many people were close to the whiteboard is used to determine when to start the audio and notes recording. During the access phase, participants can use context information such as the location of the meeting, time and date of the meeting, and the names of participants at the meeting to retrieve the recorded meeting information. This extra context makes it easier for participants to retrieve the meeting information at a later date.

Again, the details of the widget are kept transparent from the programmer. The programmer, another member of our research group, simply needed to determine which widgets he was interested in and handle the information those widgets were providing. In all, the application only required changing/adding 25 lines of Java code (out of a total of 892 lines) and modifications were localized in a single class file. The significant modifications include 2 lines added to use the context toolkit and widget, 1 line modified to enable the class to handle callbacks, and 17 lines that are application specific. Comparatively, the size of the context toolkit is 5498 lines of Java code.

To achieve such easy retrofitting of context handling capabilities in existing applications, the context toolkit manages the mechanics of context acquisition and abstraction. We now turn to the relevant implementation details of these mechanics.

IMPORTANT IMPLEMENTATION DETAILS

We have previously outlined the application programmer's interface to the context toolkit by describing what is a context widget, giving some examples of widgets and demonstrating context-enabled applications that make use of the widgets. There are some important requirements (points 3 and 4 in the introduction) for the context toolkit having to do with its distribution, heterogeneity and dynamism that we will address in this section.

Distribution

The context infrastructure must accommodate distribution of widgets and applications across a network. Applications may require the services of several widgets distributed across different machines, as described in the DUMMBO application.

In addition, widgets themselves may be distributed. A widget may consist of any number of three types of components: generators, interpreters, and servers (see figure 2). A generator, as described earlier, acquires raw context information from hardware or software sensors and provides them to widgets.

An interpreter abstracts raw or low-level context information into higher level information. An example of this was seen in the DUMMBO application where the basic *NamePresence* widget used a generator to obtain a user ID for the user whose presence was detected. An interpreter is used to abstract the raw ID information into an actual user name. Interpreters can be used for more than simply converting between types of data. For example, the data from multiple *IdentityPresence* and *Activity* widgets could be used by an interpreter to determine that a meeting is taking place.

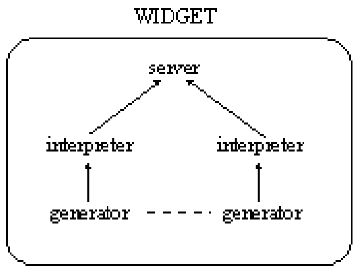


Figure 2. Anatomy of a context widget.

A server collects and stores information from interpreters and generators. While we have yet to implement a server, we believe they will be very useful. We envision using a separate server to encapsulate the information for each user, for each object, and for each physical space. This will simplify the design of applications by hiding even more complexity within the context infrastructure. For example, in the PersonFinder (extended in/out board) application, rather than have the application subscribe to every *IdentityPresence* widget in the building, it could simply subscribe to the building server and receive the same desired information.

As well, to address the privacy concerns that are raised by context-sensing, a server can be used to encapsulate a privacy manager for its given domain (whether it is a person, place, or thing). For example, if applications access their desired context information via servers, then users who don't want particular information made public can modify the privacy restrictions for their personal server to keep that information from anyone but themselves or a trusted group.

Communicating across heterogeneous components

With all the communication between the components that make up a widget and between widgets and applications, we needed to support a single, simple communications and language model. To allow as many systems as possible to employ our context toolkit, we only assume that the underlying system supports TCP/IP.

To further this goal of platform independence, our communication model uses the fairly ubiquitous HTTP protocol and our language model uses the ASCII-based Extensible Markup Language (XML) [19]. ASCII text is the lowest common denominator available on a wide variety of platforms for data transfer and XML allows us to describe structured data using text. Implementations of HTTP servers and clients libraries that can interpret XML are beginning to appear and have minimal resource requirements, allowing communication across a wide variety of platforms and devices.

Handling dynamism

Context information is inherently dynamic. As changes in the environment are detected, applications must be given the opportunity to easily adapt to these changes. This requires two components: providing access to the information using a standard mechanism and allowing access to only the desired information.

The communications model just described aids in allowing applications access to the changing information. To ease programmatic access to context information, we provide a standard subscription mechanism, enabling an application to be notified of context changes, and a polling mechanism, enabling an application to inquire about context information.

Context widgets allow applications to specify conditions that must be fulfilled before they will be notified of a change in the context. This shifts the task of filtering unwanted information to the context infrastructure and away from the application. An example of this can be seen in the DUMMBO application. This application needs to know where the mobile whiteboard is at all times, so it must subscribe to all *NamePresence* widgets in the building. However, it is only interested in callbacks where presence of the mobile whiteboard is detected. By specifying this condition in the subscription, the application can more efficiently deal with the information that it is particularly interested in and not have to deal with all the presence detections for other objects and people.

RELATED WORK

Previous research efforts have proposed infrastructures for context-enabled applications. We review them in this section and point out the differences with the context toolkit.

Schilit's infrastructure for ubiquitous computing is probably the earliest attempt at providing services for handling context [15]. In this work, context information is primarily location. Location is acquired from an Active Badges infrastructure. Active Map objects gather context information related to a physical spatial area and make it available to client applications. This approach assumes that a location can be assigned to all context information.

The stick-e framework addresses the needs of context-aware notes used, for example, to make up a tour guide [3]. Notes use SGML tags to register interest in context information and set conditions on context values that will trigger the display or execution of the note. Although this model is potentially wide ranging, it is mainly aimed at displaying context information or triggering simple actions. The mandatory use of notes as clients of context information makes it difficult to retrofit an existing application with context sensing or even build an application that modifies its behavior in response to a changing environment.

The Situated Computing Service (SitComp) has objectives very close to ours: it seeks to insulate applications from sensors used to acquire context [9]. A SitComp service is a single server that encapsulates context acquisition and abstraction and provides both an event-driven and a query interface to applications. The sensors used are location-tracking tags very similar to Active Badges. Our work goes one step further by advocating a modular structure made of several widgets dedicated to handling specific pieces of context information and laying the grounds for reusable context handling building blocks.

Although simply aimed at conveying context information to other users in a CSCW setting, the AROMA prototype shares an interesting feature with our context toolkit [13]. It provides "abstractor" objects that abstract high-level context information from the raw information captured by sensors.

Finally, CyberDesk was inspirational to the work presented in this paper [6]. Although it only deals with one piece of context information, namely the user's current text selection, it proposes a modular structure that separates context acquisition, context abstraction mechanisms and actual client services.

FUTURE WORK

The context toolkit we have presented still needs additional work to accommodate the wide range of context-enabled applications. Two areas we want to explore are extending the toolkit capabilities and structuring the widgets design space.

To extend the toolkit capabilities, we need to address the issues of resource discovery and storage and access of context history. Resource discovery enables applications to adapt transparently to changes in the infrastructure. Widgets, generators, or interpreters may become active or inactive, migrate from machine to machine and even modify the set of capabilities they provide. Context information may sometimes only be useful when examined over a period of time and thus requires context history to be stored, for example, to support context-based retrieval.

Work on the context widget library we have described in this paper is only in its initial stage. Although it has proved useful in its current state, the number and variety of widgets should be increased to effectively support a wide range of context-enabled applications. An immediate concern is to devise a structure for organizing widget classes in the library. Identifying the basic pieces of context information needed by applications, defining widgets to handle them and then combining these widgets will allow us to enhance the widget library and construct more diverse applications. So far, Presence, Identity, and Activity appear to be core types of context information and we plan to build upon this list.

CONCLUSION

We have presented a toolkit that supports the development of context-enabled applications. The context toolkit was inspired by the success of GUI toolkits and has similar benefits: building blocks called context widgets provide reusable solutions for context handling; by delegating details of context handling to the toolkit, we achieve separation of concerns between context sensing and application semantics.

To assess the validity of our toolkit-based approach, we have developed a small number of context widgets and example applications described in this paper. We were able to build new context-enabled applications, replicate canonical context-enabled applications found in the literature and retrofit an existing application with context capabilities.

More information on the work described in this paper is available on the web at http://www.cc.gatech.edu/fce/contexttoolkit/. The context toolkit software will be available by conference time from the same web page.

ACKNOWLEDGMENTS

To be filled in upon acceptance.

REFERENCES

1. Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R. and Pinkerton, M. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks 1997,* 3, pp. 421-433.
2. Brotherton, J.A., Abowd, G.D. and Truong, K.N. Supporting capture and access interfaces for informal and opportunistic meetings. Submitted for review to *CHI'99*, September 1998.
3. Brown, P.J. The stick-e document: a framework for creating context-aware applications. *Electronic Publishing. 9,* 1 (September 1996), 1-14
4. Brown, P.J., Bovey J.D., & Chen, X. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications, 4*, 5 (October 1997), 58-64.
5. Dallas Semiconductor Corporation. iButton Home Page. Available at http://www.ibutton.com.
6. Dey, A., Abowd, G.D. and Wood, A. CyberDesk: A framework for dynamic integration of network-based applications. *Knowledge-Based Systems*, 1998. To appear.
7. Fels, S, Sumi, Y., Etani, T., Simonet, N., Kobayshi, K. and Mase, K. Progress of C-MAP: A Context-Aware Mobile Assistant. In *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, Technical Report SS-98-02 (Palo Alto CA, March 1998), AAAI Press, 60-67.
8. Finney, J. and Davies, N. FLUMP: The FLexible Ubiquitous Monitor Project Home Page. Available at http://www.comp.lancs.ac.uk/computing/staff/joe/papers/flumpdh.html.
9. Hull, R., Neaves, P., and Bedford-Roberts, J. Towards Situated Computing. In *Proceedings of 1st International Symposium on Wearable Computers, ISWC '97* (Boston MA, October 1997), IEEE Press, 146-153.
10. Lamming, M. & Flynn, M. Forget-me-not: Intimate computing in support of human memory. In *Proceedings of FRIEND21: International Symposium on Next Generation Human Interfaces (1994)*, 125-128.
11 Lancaster University Mobile Computing Research Team. The Active Badge Tourist Application Web page. Available at http://www.comp.lancs.ac.uk/computing/research/mpg/most/abta_project.html.

12. Myers, B.A. A New Model for Handling Input. *Transactions on Office Information Systems, 8*, 3 (March 1990), 289-320.

13. Pederson, E.R. and Sokoler, T. AROMA: Abstract Representation of Presence Supporting Mutual Awareness. In *Proceedings of CHI '97* (Atlanta GA, March 1997), ACM Press, 51-58.

14. Rhodes, B.J. The Wearable Remembrance Agent: A System for Augmented Memory. In *Proceedings of 1st International Symposium on Wearable Computers, ISWC '97* (Boston MA, October 1997), IEEE Press, 123-128.

15. Schilit, W.N. System architecture for context-aware mobile computing. Ph.D. Thesis, Columbia University. May 1995.

16. Want, R., Hopper, A., Falcao, V., and Gibbons, J. The Active Badge Location System. *ACM Transactions on Information Systems. 10*, 1 (January 1992), 91-102.

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

18. Weiser, M. The Computer for the 21st Century. *Scientific American 265*, 3 (September 1991), 94-104.

19. XML Home Page. Available at http://www.w3.org/XML/.