Distributed mediation of imperfectly sensed context in aware environments

Anind K. Dey, Jennifer Mankoff and Gregory D. Abowd College of Computing & Graphics, Visualization, & Usability Center Georgia Institute of Technology Atlanta, GA 30332-0280 {anind, jmankoff, abowd}@cc.gatech.edu

ABSTRACT

Current context-aware services make the assumption that the context they are dealing with is correct. However, in reality, both sensed and interpreted context is often imperfect. In this paper, we describe an architecture that supports the building of context-aware services that assume context is imperfect and allows for the refinement of this imperfect context by mobile users in aware-environments. We discuss the architectural mechanisms and design heuristics that arise from supporting this refinement over space and time. We illustrate the use of our architecture and heuristics through two example context-aware services, an In-Out Board for the home and a situation-aware reminder tool.

KEYWORDS: context-aware computing, distributed mediation, aware environments, ubiquitous computing, mobile computing

INTRODUCTION

A characteristic of an aware environment is that it senses context, information sensed about its occupants and their activities, and reacts appropriately to this context, by providing context-aware services that facilitate the occupants in their everyday actions. Researchers have been trying to build tools and architectures to facilitate the creation of these context-aware services by providing ways to more easily acquire, represent and distribute sensed data. Our experience shows that though sensing is becoming more cost-effective and ubiquitous, it is still imperfect and will likely remain so. A challenge facing the development of realistic context-aware services, therefore, is the ability to handle imperfect context.

In this paper, we present an architectural solution for implementing context-aware services that assume imperfect information about the context of humans in mobile settings and allows for the distributed, refinement of incorrect context by humans in aware environments. This architecture allows humans in an aware environment to detect errors in sensed information about them and their intentions and correct those errors in a variety of ways. In previous work, we have presented an architecture for the development of context-aware services, but that context was assumed to be perfect [8]. We have also developed an architecture to support the mediation of errors in recognition-based interfaces [14]. The work presented here combines and expands those previous solutions to confront additional challenges that arise in highly mobile interactive environments. Imperfectly sensed context produces errors similar to recognition-based interfaces, but there are additional challenges that arise from the inherent mobility of humans in aware environments. Specifically, since users are likely to be mobile in an aware environment, the interactions necessary to alert them to possible context errors (from sensing or the interpretation of sensed information) and allow for the smooth correction of those errors must occur over some time frame and over some physical space. We discuss some of the new architectural mechanisms and heuristics that come into play, since designing correction strategies over time and space involves more than just an architecture that supports error mediation.

After summarizing some related work, we will present brief overviews of our own previous work that we have extended. We overview the Context Toolkit, an infrastructure to support the rapid development of context-aware services, which has assumed perfect context sensing in the past. We then overview OOPS (Organized Option Pruning System), an architecture for the mediation of errors in recognitionbased interfaces. We describe an extended Context Toolkit that no longer assumes perfectly sensed context and expands on mechanisms from OOPS to allow for the mediation of imperfectly-sensed context.

We then describe two separate implemented scenarios that use the extended Context Toolkit to facilitate interactions in aware environments that are sympathetic to errors. The first example, set in a home, shows how low-level identity and intention recognition can be corrected in a variety of ways. To facilitate this correction, the feedback and mediation strategies must be distributed wisely. The second example, set in an office, deals with higher-level human intentions, something that will not likely ever be automatically and perfectly sensed, and how an environment can make reasonable assumptions of these intentions and allow for the modification over time. We conclude the paper with a discussion of further challenges in context-aware computing that deal with mediating interactions over time and space.

RELATED WORK

Over the past several years, there has been a number of research efforts whose ultimate goal has been to create a ubiquitous computing environment, as described by Weiser [20]. We divide these efforts into three categories: aware environments, architectures to support context-aware services, and relevant context-aware services.

An aware environment is an environment that can automatically or *implicitly* sense information or context about its occupants and itself and can take action on this context. In the Reactive Room project, a room used for video conferencing was made aware of the context of both users and objects in the room for the purpose of relieving the user of the burden of controlling the objects [5]. For example, when a figure is placed underneath a document camera, the resulting image is displayed on a local monitor as well as on remote monitors for remote users. Similarly, when a videotape is being played, the lights in the room will automatically dim.

Along the same line, Mozer built the Neural Network House, a house that uses neural networks to balance the environmental needs of a user against the costs of heating and lighting the house [15]. The house gradually learns occupants' patterns of heating and lighting control for various locations in the house and various times of day. It uses this knowledge to predict user needs and automatically controls the heating and lighting, while, at the same time, attempting to minimize the overall expense incurred from them.

Environmental control was also a goal in the Intelligent Room project [4]. The Intelligent Room uses automatically sensed context along with explicit user input to adapt applications or services. Example services include turning off the lights and turning on soothing music when an occupant lies down on a sofa, and retrieving weather or news information that are relevant to the current context.

Bobick et al. also built an aware environment called KidsRoom [1]. KidsRoom was an interactive narrative space for children. The elements of the narrative continued based on the implicitly sensed activities of the children. These activities included talking to and dancing with avatars displayed on the walls.

These four aware environments all share the same property: they use implicit sensing but ignore any uncertainty in the sensed data and its interpretations. If the environment takes an action on incorrectly sensed input, it is the occupant's responsibility to undo the incorrect action (if this is possible) and to try again.

A number of architectures that facilitate the building of context-aware services, such as those found in aware environments, have been built [2,6,9,11,13,19]. Unfortunately, as in the case of the aware environments, a simplifying assumption is made that the context being implicitly sensing is 100% certain. Context-aware services that are built on top of these architectures act on the provided context without any knowledge that the context is potentially uncertain.

There are some exceptions to this assumption about certainty. We will examine two context-aware services that illustrate how individual services have attempted to take uncertainty of sensed input into account. The first is the Remembrance Agent, a service that examines the user's location, identity of nearby individuals, and the current time and date to retrieve relevant information [18]. The interpretation of the sensed context into relevant information is uncertain here. Rather than displaying the information with the highest calculated relevance, the Remembrance Agent instead presents the user with a list of the most relevant pieces of information and the relevance factor for each. In this way, the user can choose what is most relevant to the current situation from a filtered set.

The second service we will discuss is Multimodal Maps, a map-based application for travel planning [3]. Users can determine the distances between locations, find the location of various sites and retrieve information on interesting sites using a combination of direct manipulation, pen-based gestures, handwriting and speech input. When a user provides multimodal input to the application, the application uses multimodal fusion to increase the likelihood of recognizing the user's input. Rather than take action on the most likely input, if there is any uncertainty or ambiguity remaining after fusion, the application prompts the user for more information. By prompting the user for additional information, the system reduces the chance of making a mistake and performing an incorrect action.

Rather than assuming that sensed input (and its interpretations) is perfect, the two services demonstrate two techniques for allowing the user to correct uncertainty or ambiguity in implicitly sensed input. Note that both systems require explicit input on the part of the user before they can take any action. The goal in our work is to provide an architecture that supports a variety of techniques, ranging from implicit to explicit, that can be applied to contextaware services. By removing the simplifying and incorrect assumption that all context is certain, we are attempting to facilitate the building of more realistic context-aware services.

EXISTING SUPPORT

Similar to the architectures described in the previous section, we have built an architecture for the development of context-aware services, but the context it used was assumed to be perfect [8]. We have also developed an architecture that supports the mediation of errors in recognition-based interfaces [14]. We will now describe the relevant features of both architectures. The next section will describe how these architectures were combined and extended to support users in mediating imperfectly sensed context.

The Context Toolkit

The Context Toolkit is a software toolkit that is aimed at allowing others to build services that support mobile users in aware environments. The Context Toolkit makes it easy to add the use of context or implicit input to existing applications that don't use context.

The Context Toolkit consists of three basic building blocks: context widgets, context aggregators and context interpreters. Figure 1 shows the relationship between sample context components and applications.



Figure 1: Context Toolkit components: arrows indicate data flow.

Context widgets encapsulate information about a single piece of context, such as location or activity, for example. They provide a uniform interface to components or applications that use the context, hiding the details of the underlying context-sensing mechanism(s), allowing them to treat implicit and explicit input in the same manner. Context widgets allow the use of heterogeneous sensors that sense redundant input, regardless of whether that input is implicit or explicit. Widgets maintain a persistent record of all the context they sense. They allow other components to both poll and subscribe to the context information they maintain. Widgets are responsible for collecting information about the environment.

A context interpreter is used to abstract or interpret context. For example, a context widget may provide location context in the form of latitude and longitude, but an application may require the location in the form of a street name. A context interpreter may be used to provide this abstraction. A more complex interpreter may take context from many widgets in a conference room to infer that a meeting is taking place.

A context aggregator is very similar to a widget, in that it supports the same set of features as a widget. The difference is that an aggregator collects multiple pieces of context. In fact, it is responsible for the entire context about a particular entity (person, place, or object). Aggregation facilitates the access of context by applications that are interested in multiple pieces of context about a single entity.

Context components are intended to be persistent, running 24 hours a day, 7 days a week. They are instantiated and executed independently of each other in separate threads and on separate computing devices. The Context Toolkit makes the distribution of the context architecture transparent to context-aware applications, handling all communications between applications and components.

OOPS

OOPS is a GUI (graphical user interface) toolkit that provides support for building interfaces that make use of recognizers. Like sensing, recognition is ambiguous, and OOPS provides support for tracking and resolving, or *mediating*, uncertainty.

Because OOPS provides support for building interfaces, our focus in OOPS is on integrating recognition into the exist-

ing input dispatching system of subArctic [10], the GUI toolkit that OOPS is built upon. This means that, for example, if a recognizer produces text events, they will be dispatched through the same mechanism as characters produced by a keyboard, and thus will be available to any-thing that consumes keyboard input.

Additionally, OOPS provides an internal model of recognized input, based on the concept of hierarchical events [16], that allows separation of mediation from recognition and from the application. As we will see, this is a key abstraction that we use in the extended Context Toolkit.

Our model uses a directed graph to keep track of source events, and their interpretations (which are produced by one or more recognizers). For example, when the user speaks, a speech recognizer may take the audio (the source event) and produce sentences as interpretations. These sentences may be further interpreted, for example by a natural language system, as nouns, verbs, etc. Figure 2 shows the resulting graph.



Figure 2: Graph representing interpretations.

Note that there are two different sentences shown in this graph, at most one of which is correct (*i.e.* is what the user actually said). We call this situation *ambiguous* and mediation is used to resolve the ambiguity. In particular, a mediator will display feedback about one or more interpretations to the user, who will then select one or repeat her input.

Once the correct input is known, OOPS updates the directed graph to include information about which events were accepted and rejected, and notifies the recognizer that produced the events and any consumers of the events, of what happened. At this point, consumers can act on an event (for example, by executing the command specified by the user in his speech).

To summarize, OOPS automatically identifies ambiguity in input and intervenes between the recognizer and the application by passing the directed graph to a mediator. Once the ambiguity is resolved, OOPS allows processing of the input to continue as normal.

MEDIATING IMPERFECT CONTEXT

As stated previously, the Context Toolkit consists of widgets that implicitly sense context, aggregators that collect related context, interpreters that convert between context types, applications that use context and a communications infrastructure that delivers context to these distributed components. OOPS consists of applications (interfaces) that produce input, recognizers that convert between input types and applications that use input. We will now discuss how the Context Toolkit and OOPS were combined and extended to support context-aware services that can deal with ambiguous context.

In order to understand our extensions, consider a single interaction. Initially, context is implicitly sensed by a context widget. This context is sent to an interpreter that is equivalent to a recognizer in OOPS. A context interpreter now creates multiple ambiguous interpretations of that context. Each interpretation is an event that contains a set of attribute name-value pairs (a piece of context) and information about what it is an interpretation of (its source) and who produced it. The result is a directed graph, just like the representation used in OOPS.

Once the widget receives the interpretations, it sends them to all of its subscribers. Note that the widget does not send the entire graph to subscribers, just those events that match the subscription request. This is done in order to minimize network calls (which can be quite extensive and costly) as a graph is generated and mediated.

Since widgets and aggregators, as well as applications, may subscribe to a widget, all of these components must now include support for dealing with ambiguous events (and mediation of those events). A subscriber in the Context Toolkit receives data through an input handler, which is responsible for dealing with distributed communications. This input handler checks incoming context for ambiguity, and, if necessary, sends the context to a mediator instead of the subscribing component. Mediators intervene between widgets (and aggregators) and applications in the same way that they intervene between recognizers and applications in OOPS. Since the subscriber may be an aggregator or a widget, there is a need for distributed feedback services that are separate from applications. For example, a mediator may use the widget that generated the ambiguous event to communicate with the user (since sensors are usually co-located with the things that they are sensing).

Once a mediator provides feedback to the user, the user responds with additional input. The mediator uses this information to update the event graph. It does this by telling the widget that produced the events to accept or reject them as correct or incorrect. The widget then propagates the changes to its subscribers. If ambiguity is resolved, the events are delivered as normal and subscribers can act on them. Otherwise, the mediation process continues.

Summary

We have illustrated the basic architecture of our extended toolkit. The implications of adding mechanisms from OOPS to the Context Toolkit are quite large. Even though the high-level abstractions are similar, in practice the extended Context Toolkit has to deal with new issues that were not relevant to the design of OOPS. First, because the Context Toolkit is a distributed system and because mediation is an interactive process that requires appropriate response times, only those portions of the event graph that are subscribed to are passed across the network. No single component can contain the entire graph being used to represent ambiguity. Providing each widget, aggregator and application with access to the entire graph for each piece of context and having to update each one whenever a change occurred (new event is added or an event is accepted or rejected) impedes the system's ability to deliver context in a timely fashion, as is required to provide feedback and action on context.

Second, because input may be produced in many different physical locations, the architecture supports distributed feedback. This allows mediation to occur in the user's location (where the input was sensed). To support distributed feedback, we have extended context widgets to support feedback and action via output services. Output services are quite generic and can range from sending a message to a user to rendering some output to a screen to changing the connections between context components to changing the appearance of the environment. For example, we have some output services that send email or text messages to arbitrary display devices and others that can control appliances such as lights and televisions.

EXPLORING DISTRIBUTED MEDIATION IN PRACTICE

In the previous section, we described modifications to the Context Toolkit that will allow for human driven distributed mediation of imperfectly sensed and interpreted context. In this section, we want to demonstrate how the architectural solutions provided by the modified Context Toolkit are put into practice in more realistic aware environment interactions. We will explore two different settings, a home and an office, and two different forms of context, low-level identification and higher level intention. What we will show in these two examples is not only the specifics of applying the modified Context Toolkit, but a demonstration of important heuristics that go beyond what an architecture can provide and which come up in designing distributed mediation when mobility is involved. Briefly, these heuristics fall into 3 categories:

Providing redundant mediation techniques: One of the attractive features of context-aware computing is the promise that it will allow humans to carry out their everyday tasks without having to provide additional explicit cues to some computational service. Our experience shows, however, that the more implicit the gathering of context, the more likely it is to be in error. In designing mediation techniques for correcting context, a variety of redundant techniques should be provided simultaneously. This redundant set not only provides a choice on the form of user input and system feedback, but also the relative positioning and accessibility to the user should be carefully thought out to provide a smooth transition from most implicit (and presumably least obtrusive) to the most explicit [17].

Spatio-temporal relationship of input and output: Some

input must be sensed before any interpretation and subsequent mediation can occur. Because we are assuming user mobility, this means that the spatial relationship of initial input sensors must mesh with the temporal constraints to interpret that sensed input before providing initial feedback to the user. Should the user determine that some mediation is necessary, that feedback needs to be located within range of the sensing technologies used to mediate the context. Mediating interactions should occur along the natural path that the user would take. In some cases, this might require duplicate sensing technologies to take into account different initial directions in which a user may be walking. In addition, the mediation techniques may need to have a carefully calculated timeout period, after which mediation is assumed not to happen.

Effective use of defaults: Sometimes the most effective and pleasurable interactions are ones that do not have to happen. Prudent choice of default interpretations can result in default mediated actions that occur when no additional correction is provided by the user. These defaults could either provide some default action or provide no action, based on the situation.

Example 1: Mediating simple identity and intention in the Aware Home

The first service that we will describe is an In-Out Board installed in a home. The purpose of this service is to allow occupants of the home and others (who are authorized) outside the home to know who is currently in the home and when each occupant was last in the home. This service may be used by numerous other applications as well. It is a piece of the Georgia Tech Broadband Residential Laboratory, a house that is being instrumented to be a context-aware environment [12].

Physical Setup

Occupants of the home are detected when they arrive and leave through the front door, and their state on the In-Out Board is updated accordingly. Figure 3 shows the front door area of our instrumented home, taken from the living room. In the photographs, we can see a small anteroom with a front door and a coat rack. The anteroom opens up into the living room, where there is a key rack and a small table for holding mail – all typical artifacts near a front door. To this, we have added two ceiling-mounted motion detectors (one inside the house and one outside), a display, a microphone, speakers, a keyboard and a dock beside the key rack.

When an individual enters the home, the motion detectors detect his presence. The current time, the order in which the motion detectors were set off and historical information about people entering and leaving the home is used to infer who the likely individual is and whether he is entering or leaving. This inference is indicated to the person through a synthesized voice that says "Hello Jen Mankoff" or "Goodbye Anind Dey", for example. In addition, the wall display shows a transparent graphical overlay (see figure 4) that indicates the current state and how the user can correct it if it is wrong: speak, dock or type. If the inference is correct, the individual can simply continue on as usual and the In-Out Board display will be updated with this new information.



Figure 3: Photographs of In-Out Board physical setup.

😸 Who's home?	
Goodbye Gregory Abowd	Anind Dey Out 5:59pm
Please dock, speak, or type Renata Fortes if this is wrong	Tanisha Hall Out 5:59pm
Cory Kidd Out 5:59pm	Kent Lyons Out 5:59pm
Jen Mankoff Out 5:59pm	David Nguyen Out 5:59pm
Rob Orr Out 5:59pm	Daniel Salber Out 5:59pm
Brad Singletary Out 5:59pm	Randy and Steve Out 5:59pm
Khai Truong Out 5:59pm	Gregory Abowd Out 5:59pm
Enter your name in	

Figure 4: In-Out Board with transparent graphical feedback.

If the inference is incorrect, the individual has a number of ways to correct the error. First, let us point out that the inference can be incorrect in different ways. The direction, identity or both may be wrong. The individual can correct the inference using a combination of speech input, docking with an iButton, and keyboard input on the display. These three input techniques plus the motion detectors range from being completely implicit to extremely explicit. Each of these techniques can be used either alone, or in concert with one of the other techniques. After each refinement, additional feedback is given indicating how the new information is assimilated. There is no pre-defined order to their use. Changes can continue to be made indefinitely, however, if the user makes no change for a pre-defined amount of time, mediation is considered to be complete and the service updates the wall display with the corrected input. The timeout for this interaction is set to 20 seconds.

For example, the user can say "No", "No, I'm leaving/arriving", "No, it's Anind Dey", or "No, it's Anind Dey and I'm arriving/leaving". The speech recognition is not assumed to be 100% accurate, so the system again indicates its updated understanding of the current situation via synthesized speech.

Alternatively, an occupant can dock her iButton. An iButton is a button that contains a unique id that the system uses to determine the identity of the occupant. The system then makes an informed guess based on historical information as to whether the user is coming or going. The user can further refine this using any of the techniques described if it is wrong.

Finally, the occupant can use the keyboard to correct the input. By typing his name and a new state, the system's understanding of the current situation is updated.

Architecture

We will now discuss how the architecture facilitated this service. The following figure (Figure 5) shows the block diagram of the components in this system.



Figure 5: Architecture diagram for In-Out Board. There are 4 widgets providing context, two of which have output services for feedback: speech output and visual feedback display.

Input is captured via context widgets that detect presence, using either the motion detectors, speech recognition, iButton or keyboard as the input-sensing mechanism. All of these widgets existed in the original Context Toolkit, but were modified to be able to generate ambiguous as well as unambiguous context information.

The motion detector-based widget uses an interpreter to interpret motion information into user identity and direction. The interpreter uses historical information collected about occupants of the house, in particular, the times at which each occupant has entered and left the house on each day of the week. This information is combined with the time when the motion detectors were fired and the order in which they were fired. A nearest-neighbor algorithm is then used to infer identity and direction of the occupant. The speech recognition-based widget uses a pre-defined grammar to determine identity and direction.

When any of these widgets capture input, they produce not only their best guess as to the current situation, but also likely alternatives as well, creating an ambiguous event graph. The wall display has subscribed to unambiguous context information and is not interested in ambiguous information. When ambiguous information arrives, it is intercepted by a mediator that resolves it so that it can be sent to the application in its unambiguous form. The mediator uses this ambiguous information to mediate (accept and reject) or refine alternatives in the graph. The entire ambiguous graph is not held by any one component. Instead, it is distributed among the four context widgets and the mediator. Each component can obtain access to the entire graph, but it is not necessary in this service.

The mediator creates a timer to create temporal boundaries on this interaction. The timer is reset if additional input is sensed before it runs out. As the mediator collects input from the user and updates the graph to reflect the most likely alternative, it provides feedback to the user. It does this in two ways. The first method is to use a generic output service provided by the Context Toolkit. This service uses IBM ViaVoiceTM to produce synthesized speech to provide feedback to the user. The second method is applicationspecific and is the transparent graphical overlay on the wall display shown in Figure 4. The transparent overlay indicates what the most likely interpretation of the user's status is and what the user can do to change their status: e.g. "Hello Anind Dey. Please dock, type, or speak if this is wrong." As the timer counts down, the overlay becomes more transparent and fades away.

When all the ambiguity has been resolved in the event graph and the timer has expired, the overlay will be faded completely and the correct unambiguous input is delivered to the wall display and the display updates itself with the new status of the occupant. Also, the input is delivered back to the interpreter so it has access to the updated historical information to improve its ability to infer identity and direction.

Design Issues

In this section, we will further investigate the design heuristics, introduced in a previous section, that arose during the development of this service. The first issue is how to supply redundant mediation techniques. On the input side, in an attempt to provide a smooth transition from implicit to explicit input techniques, we chose motion detectors, speech, docking and typing. In order to enter or leave the house, users must pass through the doorway, so motion detectors are an obvious choice to detect this activity. Often users will have their hands full, so speech recognition is added as a form of more explicit, hands-free input. iButton docking provides an explicit input mechanism that is useful if the environment is noisy. Finally, keyboard input is provided as an additional explicit mechanism and to support the on-the-fly addition of new occupants and visitors.

A valid question to ask is why not use sensors that are more accurate. Unfortunately in practice, due to both social and technological issues, there are no sensors that are both reliable and appropriate. As long as there is a chance that the sensors may make a mistake, we need to provide the home occupants with techniques for correcting these mistakes. None of the sensors we chose are foolproof either, but the combination of all the sensors and the ability to correct errors before applications take action is a satisfactory alternative.

On the output side, synthesized speech is used both to mirror the speech recognition input and to provide an output mechanism that is accessible (i.e. audible) to the user throughout the entire interaction space. Visual output for feedback is provided in the case of a noisy environment and for action as a persistent record of the occupancy state.

The next design decision is where to place the input sensors and the rendering of the output to address the spatiotemporal characteristics of the physical space being used. There are "natural" interaction places in this space, where the user is likely to pause: the door, the coat rack, the key rack and the mail table. The input sensors were placed in these locations: motion sensors on the door, microphone in the coat rack, iButton dock beside the key rack and keyboard in a drawer in the mail table. The microphone being used is not high quality and requires the user to be quite close to the microphone when speaking. Therefore the microphone is placed in the coat rack where the user is likely to be leaning into when hanging up their coat. A user's iButton is carried on the user's key chain, so the dock is placed next to the key rack. The speakers for output are placed between the two interaction areas to allow it to be heard throughout the interaction space. The display is placed above the mail table so it will be visible to individuals in the living room and provide visual feedback to occupants using the iButton dock and keyboard.

Another design issue is what defaults to provide to minimize required user effort. We use initial feedback to indicate to the user that there is ambiguity in the interpreted input. Then, we leave it up to the user to decide whether to mediate or not. The default is set to the most likely interpretation, as returned by the interpreter. Through the use of the timeout, the user is not forced to confirm correct input and can carry out their normal activities. This is to support the idea that the least effort should be expended for the most likely action. The length of the timeout, 20 seconds, was chosen to allow enough time for a user to move through the interaction space, while being short enough to minimize between-user interactions.

We added the ability to deal with ambiguous context, in an attempt to make these types of applications more realistic. Part of addressing this realism is dealing with situations that may not occur in a prototype research environment, but do occur in the real world. An example of this situation is the existence of visitors, or people not known to the service. To deal with visitors, we assume that they are friendly to the system, a safe assumption in the home setting. That means they are willing to perform minimal activity to help keep the system in a valid state. When a visitor enters the home, the service provides feedback (obviously incorrect) about who it thinks this person is. The visitor can either just say "No" to remove all possible alternatives from the ambiguity graph and cause no change to the display, or can type in their name and state using the keyboard and add themselves to the display.

Example 2: Mediating higher level intention with an office reminder system

The second service that we will describe is CybreMinder, a situation-aware reminder tool [7]. It allows users to create a reminder message for themselves or someone else and to associate a situation with it. The reminder message will be delivered when the associated situation has been satisfied. The purpose of this tool is to trigger and deliver reminders at more appropriate times than is currently possible.

Physical Setup

Various locations (building entrances and offices) in two buildings have been instrumented with iButton docks to determine the location of building occupants. With each dock is a computer screen on which reminder messages can be displayed. Figure 6 shows an example installation.



Figure 6: Reminder display with iButton dock.

Interaction

Users can create situation-aware reminders on any networked device. We will illustrate the interaction through the use of a concrete example. Jen and Anind are working on a paper for UIST 2000. Jen sent out a draft of the paper and is waiting for comments. She creates a reminder message for Anind to drop off his comments. She sets the situation in which to deliver the reminder to be when Anind enters the building.

When Anind enters the building (sensed by the appropriate iButton dock), the reminder is delivered to him on the local display (Figure 7). Just because the reminder was delivered to Anind, does not mean that he will complete the action detailed in the reminder. In fact, the most likely occurrence in this setting is that a reminder will be put off until a later time. The default status of this reminder is set to "still pending". This means that the reminder will be delivered again, the next time Anind enters the building. However, if Anind does enter Jen's office within a pre-defined amount of time, the system changes the previous incorrect reminder status to "completed" (Figure 8a). Of course, if he was just stopping by to say hello, he can dock again to return this back to "still pending" (Figure 8b). Alternatively, Anind can use a separate application, which displays all of his reminders, to explicitly change the status of a reminder.

🛱 Reminder: Comments on the UIST draft 🛛 🗖 🗙	
To:	Anind Dey
Subject:	Comments on the UIST draft
From:	Anind Dey
Priority:	Highest
	Please drop off your comments on
	the UIST paper. I'd like to start working
Message:	on the next version soon.
	jen
Situation:	username=Anind Dey, location=CRB

Figure 7: Reminder message delivered in appropriate situation.



Figure 8: Graphical feedback for reminder status. (a) shows the feedback for a "delivered" status and (b) shows the feedback for a "pending" status.

Architecture

We will now discuss how the architecture facilitated this service. The following figure (Figure 9) shows the block diagram of the components in this system.



Figure 9: Architecture diagram for reminder service. There are a number of widgets, one for each "interesting" location in the building. Each widget has two services, one for displaying a reminder and one for providing feedback about the reminder's status.

Input is captured via context widgets that detect presence, using iButtons as the input-sensing mechanism. When the information from these widgets matches a situation for which there is a reminder, the reminder is delivered to a display closest to the recipient's location. The reminder is displayed using an output service that the widgets provide. Initially, input in this service was treated as unambiguous in [7]. Using the combination of the Context Toolkit and OOPS, we have added the ability to deal with ambiguity. Now, a pair of ambiguous events is created by the widget, one indicating that the reminder is still pending and one indicating that the reminder has been completed.

The reminder service has subscribed for unambiguous context, so when the ambiguous events arrive, they are delivered to the mediator for the service. When the mediator receives this input, it creates a timer to enforce temporal boundaries on this interaction. The timer has a timeout value of 10 minutes for this service. If the user does not address and complete the reminder, the timer times out and the most likely event is chosen, that of keeping the reminder status as "still pending". If the user does address the reminder, he can dock his iButton to indicate this. This docking event is delivered to the mediator which swaps which reminder status is most likely.

Feedback is provided to the user via an audio cue and on the display that is closest to the user's current location (Figure 8) using another output service provided by the local widget. Each additional dock swaps the reminder status as well and produces feedback for the user. Timely delivery of docking events from the widget to the mediator and back to the widget to provide feedback is essential for providing the user with timely feedback. When the timer expires, the most likely reminder status is passed to the service which updates the reminder accordingly.

Design Issues

In this section, we will further investigate some of the interesting design issues that arose during the development of this service. In this service, input is provided to the system using iButtons and docks, which comprise the location system in our research building. The existing infrastructure was leveraged for this service, both for simplicity of development and to leverage off of user's knowledge about the system. Additionally, users can explicitly correct the status of a reminder using a separate interface. Output comes in the form of a simple audio cue, a beep, to get the user's attention, and a visual cue that indicates the current reminder status. Speech was not used in this setting because it was a more public space than the home.

The choice of locations for input and output was again guided by the space in which the service was deployed. Natural locations for sensing input and providing feedback were the entrances to rooms where users would naturally stop to dock anyway. If a different location system were used, the locations chosen might differ slightly. Entrances to offices would still be appropriate, as they are natural stopping places where users knock on a door. But in a conference room, the chairs where users site may be a better choice.

In this service, as opposed to the previous one, the default interpretation of user input is that no action was taken, and that the reminder is still pending. This default was chosen because the implicit user input received (a user entering the building) only causes a reminder to be delivered, and does not provide any indication as to whether the reminder has been addressed and completed. There is really no sensor or group of sensors that will enable us to accurately determine when a reminder has been addressed. We must rely on our users to indicate this. The interpretation that a reminder is still pending is the most likely interpretation and therefore it was made the default, requiring the least user action to be accepted. The timeout for accepting the input was chosen in a similar fashion as the first service, long enough to give the user an opportunity to address the reminder, while short enough to minimize overlap between individual interactions (reminders).

When designing this service, we chose to address the ambiguity only at the level of whether the reminder was dealt with or not. This was done in order to make the design simpler for demonstration purposes. The underlying context that is used to determine when a message should be delivered will also have ambiguous alternatives that may need mediation. It should not be hard to see how we could combine the type of service we demonstrated with the In-Out Board with this reminder service, to make this possible.

FUTURE WORK

The extended Context Toolkit supports the building of more realistic context-aware services, that are able to make use of imperfect context. But, we have not yet addressed all the issues raised by this problem.

Because multiple components may subscribe to the same ambiguous events, mediation may actually simultaneously in these components. When one component successfully mediates the events, the other components need to be notified. We have already added the ability for input handlers to keep track of what is being mediated locally in order to inform mediators when they have been pre-empted. However, we still need to add a priority system that will allow mediators to have some control over the global mediation process.

An additional issue we need to further explore is how events from different interactions can be separated and handled. For example, in the In-Out Board service, it is assumed that only one user is mediating their occupancy status at any one time. If two people enter together, we need to determine which input event belongs to which user in order to keep the mediation processes separate.

Finally, we need to build more context-aware services using this new architecture and put them into extended use. This will lead to both a better understanding of how users deal with having to mediate their implicit input and a better understanding of the design heuristics involved in building these context-aware services.

CONCLUSIONS

The extended Context Toolkit supports the building of realistic context-aware services, ones that deal with imperfect context and allow users to mediate the context. When users are mobile in an aware environment, the mediation is distributed over both space and time. The toolkit extends the original Context Toolkit and OOPS, providing support for the timely delivery of context via partial delivery of the event graph and distributed feedback via output services in context widgets. We introduced design heuristics that play a role in the building of distributed context-aware services. We demonstrated the use of the extended Context Toolkit and the design heuristics through two example contextaware services, an In-Out Board for the home and a situation-aware reminder system.

ACKNOWLEDGMENTS

This work was supported in part by a NSF CAREER Grant # 9703384 and a Motorola University Partnerships in Research grant.

REFERENCES

- 1. Bobick, A. et *al*. The KidsRoom: A perceptually-based interactive and immersive story environment. PRESENCE: Teleoperators and Virtual Environments, 8(4), 1999, pp. 367-391.
- Brown, P.J. The stick-e document: A framework for creating context-aware applications, in Proceedings of EP '96.
- Cheyer, A. & Julia, L. Multimodal maps: An agentbased approach. In Proceedings of the International Conference on Cooperative Multimodal Communication (CMC '95), May 1995.
- 4. Coen, M. The future of human-computer interaction or how I learned to stop worrying and love my intelligent room. IEEE Intelligent Systems 14(2), 1999, p. 8-10.
- Cooperstock, J., Fels, S., Buxton, W. & Smith, K. Reactive environments: Throwing away your keyboard and mouse, CACM 40(9), 1997, pp. 65-73.
- Davies, N., Wade, S.P., Friday, A. & Blair, G.S. Limbo: A tuple space based platform for adaptive mobile applications, in Proceedings of Conference on Open Distributed Processing/Distributed Platforms, (ICODP '97).
- 7. Dey, A.K. & Abowd, G.D. CybreMinder: A Context-Aware System for Supporting Reminders. In submission.
- Dey, A.K., Salber, D., & Abowd, G.D. A Context-based infrastructure for smart environments. In Proceedings of the International Workshop on Managing Interactions in Smart Environments (MANSE '99), pp. 114-128.
- 9. Harter, A. et al. The Anatomy of a Context-Aware Application. In Proceedings of Mobicom '99.
- Hudson, S. and Smith, I. Supporting dynamic downloadable appearances in an extensible user interface toolkit. In Proceedings of the Symposium on User Interface Software and Technology, (UIST '97), pp. 159-168.
- Hull, R., Neaves, P. & Bedford-Roberts, J. Towards situated computing. In Proceedings of 1st International Symposium on Wearable Computers (ISWC '97).
- 12. Kidd, C.D. et al. The Aware Home: A living laboratory for ubiquitous computing research. In Proceedings of

the Second International Workshop on Cooperative Buildings, (CoBuild '99).

- 13. Korteum, G., Segall, Z. & Bauer, M. Context-aware, adaptive wearable computers as remote interfaces to 'intelligent' environments, in Proceedings of 2nd International Symposium on Wearable Computers (ISWC '98), pp. 58-65.
- Mankoff, J., Hudson, S.E. & Abowd, G.D. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In Proceedings of CHI 2000, pp. 368-375.
- 15. Mozer, M. C. The neural network house: An environment that adapts to its inhabitants. In Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments, pp. 110-114.

- Myers, B.A. and Kosbie, D.S. Reusable hierarchical command objects. In Proceedings of CHI '96, pp. 260-267.
- Rhodes, B. Margin Notes: Building a contextually aware associative memory. In Proceedings of the International Conference on Intelligent User Interfaces (IUI '00).
- Rhodes, B. The Wearable Remembrance Agent: A system for augmented memory Personal Technologies (1997) 1(1), pp. 218-224.
- Schilit, W.N., System architecture for context-aware mobile computing, Ph.D. Thesis, Columbia University, May 1995.
- 20. Weiser, M. The computer for the 21st century. Scientific American 265(3), 1991, pp. 66-75.