A Peripheral Display Toolkit

Tara Matthews[1], *Tye Rattenbury*[1], *Scott Carter*[1], *Anind K. Dey*[2] *and Jennifer Mankoff*[1]

[1]EECS Department
UC Berkeley
Berkeley, CA 94720-1770, USA
ambient@guir.berkeley.edu

[2]Intel-Berkeley Research Lab
2150 Shattuck Ave, Suite 1300
Berkeley, CA 94704, USA
anind@intel-research.net

ABSTRACT

Traditionally, computer interfaces have been confined to conventional displays and focused activities. However, as displays become embedded throughout our environment and daily lives, increasing numbers of them must operate on the periphery of our attention. *Peripheral displays*, ubiquitous computing devices that present information without demanding attention, are difficult to build, particularly because they must dynamically manage the cognitive load they place on users. We present a toolkit that aids the development of peripheral displays. We determined three key issues for the toolkit, based on a survey of existing peripheral displays and cognitive science literature: abstraction of data, selection of notification levels, and transitions between notification levels. Our contribution is the investigation of these key characteristics, combined with a toolkit that encapsulates them and supports the design of displays that focus on these issues. We describe our toolkit architecture, and present five sample peripheral displays demonstrating our toolkit's capabilities.

KEYWORDS: Toolkits, ambient and peripheral displays

INTRODUCTION

Traditionally, computer interfaces have been confined to task-focused, desktop computing activities. This puts a large amount of information on a single computer screen, demanding a person's full attention. Increasingly, however, computer interfaces are moving towards a more diverse assortment of computerized devices that have many different forms of input and output [24]. These devices, referred to as ubiquitous computing devices, are meant to integrate seamlessly into the world and almost disappear [34]. However, the goal of making technology invisible has yet to be accomplished.

We present a toolkit to support the creation of applications within a subset of ubiquitous computing, called *peripheral displays*. Peripheral displays are ubiquitous computing devices that give information to a user without demanding their full attention. This allows a person to be aware of more information without being overburdened by it [33].

SUBMITTED TO UIST 2003

Peripheral displays "require minimal attention and cognitive effort and are thus more easily integrated into a persistent physical space" [1].

What does a typical peripheral display look like? It may be physical, audible, or simply displayed on a monitor. Direct interaction occurs rarely, if at all, with peripheral displays. Its data source is predominantly of low to medium importance and is continually changing. A user generally wishes to monitor this data peripherally while performing a separate primary task. She may wish to be notified when more important data arrives.

As an example of a physical peripheral display, consider the bus arrival display shown in Figure 1. Each column of LEDs indicates the distance of a bus line from the nearest bus stop, based on data published by the bus company. The LEDs can flash brightly to notify the user when a bus is close. Otherwise, they turn on one by one, indicating distance (more LEDS implies proximity) without grabbing the user's attention.

It is difficult to build such a display for several reasons. First, they are often physically-based and distributed, requiring hardware and networking, as well as software skills, and cannot be built using the tools available to traditional interface designers. These issues have been partially addressed in recent years by tools such as Phidgets [6] and iStuff [2]. Second, the key characteristics of peripheral displays (discussed in detail later in the paper), such as the selection of notification levels representing the importance of information, and the development of varied transitions for capturing different levels of attention, must be dealt with in an ad-hoc manner.

We believe there is a need for tools supporting the creation of peripheral displays. To address this need, we have designed and implemented the Peripheral Displays Toolkit

Figure 1: A display of bus arrival information.

(PTK). The PTK provides architectural support for key features of peripheral displays, allowing designers to more easily prototype them and supporting reuse of code.

Our architecture adds support for three key characteristics of peripheral displays: abstraction, notification, and transitions. *Abstraction* is used to transform incoming data to meet the needs of the output device. A designer can indicate the *notification* level of incoming data. *Transitions* are used to update the display, or output (a more neutral term that includes non-visual displays of information), to attract an appropriate amount of attention on the basis of the notification level..

The design of our toolkit is based upon cognitive science literature and an inspection of existing peripheral displays. Additionally, we designed it to support issues common to many ubiquitous computing applications: remotely distributed applications, the use of physical components, sensors, and other hardware, and the extreme diversity of input sources and output devices.

Overview

The next section presents a survey of existing peripheral displays and literature on attention, justifying the three issues supported by our toolkit (abstraction, notification, and transitions). We then present our architecture, describing how we support each of these issues. We have built five applications using the toolkit, and they are presented as illustrations of how the toolkit works. We then touch on related work in toolkit development, and close with future work and conclusions.

SURVEY OF PERIPHERAL DISPLAYS AND ATTENTION

Before beginning our survey, we need to define what is meant by the term "peripheral display." For our purposes, peripheral displays are displays that are not at the focus of a person's attention. This naturally leads to the question, what is attention?

To answer this question, we turn to cognitive science literature. Mack and Rock characterize attention as a subject's intent and expectation towards a stimulus [17]. According to recent models of human attention [28,32,18] and Rensink's Coherency Theory [30] the brain processes sensory input hierarchically. Although research on human attention is still in flux (new categories and models of attention are still being developed [15, 30]), one can categorize attention into four main zones: preattention, inattention, divided attention, and focused attention (see Figure 2). Early processing (*preattention*) handles objects without any referential frame. These objects are not inherently available for later processing and thus do not affect awareness. At the *inattention* stage, a person is not conscious of a perceptual stimulus, but the information may effect behavior [6]. The nature of inattention is hotly debated, and no rigorous definition exists. *Divided attention* and *focused attention* represent the two ways that humans consciously perceive stimuli – by using all attentional resources to focus on one stimuli, or by distributing that attention over several objects. According to Treisman, there is a "continuum between divided and focused" attention [32], and the sloping graph in Figure 2 reflects this property.

We say that a person is *aware* of a stimulus if it in some way influences behavior (*e.g.* percolates beyond the visual cortex into prefrontal planning). *Peripheral displays, then, are displays that show information that a person is aware of, but not focused on.* This includes inattention and divided attention, but not pre-attention or focused attention.

We can characterize different categories of peripheral displays found in the literature based on the degree of attention they require. Displays with change blind aspects such as AROMA and the Agentk tickers [21] make use of inattention. As an example, the Agentk tickers display changes unnoticeably by fading text. Techniques for change blind display are described in detail by Intille [12]. Ambient displays rely on divided attention. They support monitoring and remain on the periphery of a user's attention, showing information of low to medium importance. Alerting displays, such as our bus arrival display, also rely on divided attention. They remain on the periphery at most times, but may grab attention as more important information arrives. In terms of our graph (Figure 2), ambient displays might be defined as those that are "minimally attended" (*e.g.* just salient enough for conscious perception) while alerting displays are "maximally divided" (*e.g.* slightly less salient than focal tasks).

Characteristics of Peripheral Displays

We have combined our understanding of cognitive science with a survey of existing peripheral displays in order to identify the key characteristics of peripheral displays that a toolkit should support. Though we can identify key features of peripheral displays, there are few evaluations of peripheral displays (due to the difficulty of creating such

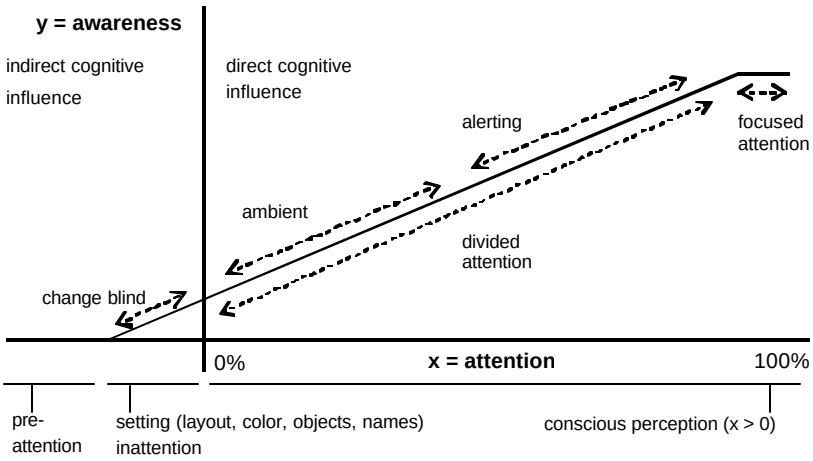

Figure 2: Graph of human awareness *vs.* attention.

Notification	**Ignore**: Possibly all display ignore some data.
	Change Blind: AROMA, Digital Family Portrait, *InfoCanvas*, Agentk Tickers
	Make Aware: AROMA, Lumitouch, Audio Aura, InfoCanvas, Pinwheels, Dangling String, Sideshow, Agentk Tickers, Kimura, *Information Percolator*, *Informative Art*
	Interrupt: Lumitouch, Sideshow
	Demand Attention: No displays from survey. Example: alarm clock.
Transition	**Explicitly supported transitions**, by notification level (Ignore and Demand Attention were not found in our survey):
	Change Blind: AROMA (animated image changes), Agentk Tickers (text fade, roll, ticker), *InfoCanvas* (may animate some image movement), Information Percolator (natural animation of bubbles moving up)
	Make Aware: Audio Aura (meaningful sounds overlaid on background sound)
	Interrupt: Sideshow (text box with important information appears), Lumitouch (LEDs flash or change color)
Abstraction	**Degradation**: Kimura (low resolution images of documents shown in a history montage of desktop activity)
	Feature abstraction: AROMA (activity), Audio Aura (email, group pulse), Digital Family Portrait (health, environment, activity, relationships), Cook's Collage (duration, heat), Lumitouch (presence, interaction), Pinwheels (LAN traffic), InfoCanvas (various), Dangling String (LAN traffic), Information Percolator (various), Informative Art (various)

Table 1: A survey of some prototypical peripheral displays, showing how they use Notification, Transition, and Abstraction. Displays include AROMA [27], the Digital Family Portrait [25], Audio Aura [26], Cook's Collage [31], Lumitouch [4], Pinwheels [5], InfoCanvas [23], Information Percolator [8], Dangling String [33], Sideshow [3], Agentk Tickers [21], Informative Art [29], and Kimura [16]. Systems in *italics* lack detail for a definitive classification. Our notification classifications are best guesses due to lack of reported evaluations.

displays) upon which to base an understanding of how they impact attention [20]. It is for this reason that we turn to cognitive science literature as additional support for the key characteristics we have identified. Our survey showed that peripheral displays have three common characteristics. These are: abstraction, notification, and transitions. This section describes these key characteristics and why they are crucial to peripheral displays. Table 1 gives examples of each key feature from our survey.

Notification

In typical use, peripheral displays allow people to monitor continually changing, non-critical data while performing a separate primary task, and to be notified when more important data arrives. So, peripheral displays can present both critical and non-critical information. It follows that critical and non-critical information must be treated differently by a display: the most critical information should be displayed so that it grabs the user's attention and potentially requires action and the least critical information should be displayed so that it does not attract conscious attention. We call these differences *notification levels*. Higher notification levels correspond to more critical data and are displayed in a way to grab attention. Lower notification levels correspond to non-critical data and are displayed to only grab peripheral attention.

Based on our survey, and the discussion of attention presented above, we defined five notification levels. Notification levels represent levels of importance. This is based both on the information source, and, ideally, some context about the interruptibility of the person receiving the information. They are "demand action," "interrupt," "make aware," "change blind," and "ignore." "Ignore" represents information that should not be displayed, and does not correspond to any attention level. "Change blind" corresponds to inattention, while "make aware" and "interrupt" correspond to a form of divided attention. "Interrupt" could also be characterized as an attempt to grab focused attention. "Demand action" is similar to "interrupt," but it also requires that the user perform some action to stop the alerting. This level, while common in GUIs, should be used sparingly in peripheral displays, because only very critical information should require the user to drop everything and attend to the displayed information. Thus, change blind displays show information with care to not distract the user, while ambient displays may attempt to make a person aware of information and alerting displays may use all levels.

The displays we surveyed used all but the demand action notification level. Make aware was by far most common, though several displays used change blind and interrupt.

Transitions

With notification levels defined, the peripheral display developer must determine how to display information to grab the appropriate amount of attention from the user. Transitions are based upon the notification level of the data, context such as the current noise level in a room, and the modality of the display. For example, if the last bus were about to arrive at a bus stop causing a notification event at the "interrupt" level, our bus arrival display might flash all of its LEDs rapidly.

Recent studies give us some guidance about how to display information corresponding to each notification level, although much work remains to evaluate exactly how subtle

or abrupt changes on a display must be to correctly grab human attention. Alerting displays typically utilize abrupt transitions for important information. Several applications [3, 4] have shown that significant changes in the interface will draw a user's attention. For ambient displays, McCrickard and Zhao found that animations like fading, rolling, and tickering made it difficult to tell when data changed [21], suggesting that repetitive and gradual animations are appropriate for change blind transitions. However, Maglio and Campbell found that continuous movement in tickers is more distracting than discrete scrolling [19]. Further research is needed to determine the best way to transition changed data in peripheral displays.

Based on these results, it is clear that animations of different types are a key tool for supporting transitions in applications that do not want to distract users. Our survey confirmed that applications explicitly supporting transitions to minimize motion are more likely to be change blind [27,21,23,8]. Other applications, such as Audio Aura, were meant to minimize distractions to the user, but did not make an effort to provide change blind transitions [26]. We categorized such displays as make aware. While in most papers, evaluations were not performed to confirm this, the motion of such applications is often significant or abrupt. Our survey also showed that alerting displays used abrupt or significant motion to purposefully interrupt [3,4].

Abstraction

Peripheral displays do not display information directly; rather they use abstraction to display information so that it may be more easily interpreted with less attention. Abstraction is the process of removing or extracting data so that the result includes fewer or different details than the original. The AROMA project showed that abstraction can convey sufficient information while remaining subdued enough to allow a user to concentrate on a main activity [27]. AROMA defined two types of abstraction: degradation and feature extraction. Degradation involves throwing away some of the original data. Feature extraction involves analyzing the original data, extracting certain features, and potentially deriving new data.

For example, in the AROMA project, remote presence was abstracted and displayed peripherally. Data in AROMA is passed through *abstractor objects* that perform basic signal processing, accumulations, and comparative analyses (such as history processing). These abstractor objects take sensor data (*i.e.*, from a microphone or camera) and create abstractions like "activity level" in the remote location. This is a form of feature extraction that derives new information from the extracted data.

Almost all of the displays we surveyed abstracted data in some way. Kimura used degradation [16], while the others all used feature extraction. Most applications used simple abstraction, without deriving new data.

In Summary

To summarize, we have identified key characteristics of peripheral displays, and broken them down in terms of features based on cognitive science literature and past work in peripheral display design and evaluation. While this survey represents a contribution, it also provides guidelines and requirements for the tool we have built. The next section describes how our toolkit addresses each of these characteristics, and describes our toolkit architecture.

ARCHITECTURE

Before describing how the Peripheral Displays Toolkit (PTK) supports the issues described above, we introduce the basic architecture. Because peripheral displays are often physically based, and because an information source may not be physically co-located with a display, we support some standard issues addressed by other similar toolkits such as AROMA [27], Real World Interfaces [22], Phidgets [6], and iStuff [2]. In particular, we support storage of history, and distributed input and output components (as do iStuff and AROMA), and easy switching of the connections between them with the help of a discovery system (inspired by iStuff). The PTK currently provides some basic library elements that may be subclassed, such as input from web pages and microphones and Phidget output.

Our contribution is the addition of support for abstraction, notification, and transitions. AROMA has some support for abstraction, and support for distributed input and output handling, but does not support notification or transitions. To our knowledge, no toolkit supports all three. Here we present the architecture in which they sit, while the next section presents the details of how each is supported.

Distributed Input, Output, and Server

The PTK has an event-based, distributed architecture consisting of three key pieces: the input source(s), the PTK server, and the output application(s). (See Figure 3.) Because these components are decoupled from each other, they are easily reused in new peripheral displays. Multiple output applications can subscribe to a single input source via the PTK server, facilitating code reuse and allowing for easier prototyping of outputs. Inputs and outputs can be easily swapped and reconnected, via the discovery system, allowing for richer interface experimentation.

Data flows from input sources to the PTK server, which routes a data event to all the outputs that have subscribed to receive events of this type. The output application is the most complex part of the toolkit, and includes support for abstraction, notification, and transitions. It borrows parts of its structure from the event-handling infrastructure first presented in ArtKit [9] and later in SubArctic [11]. However, rather than simply delivering events to a particular display device in the output application, an event is passed first to an abstraction subsystem which converts it to another data type (if necessary), then to a notification subsystem which sets the notification level, followed by an output subsystem that selects one or more output devices. Each output device then passes the event to a transition class that determines how to display it. A static event history is kept in the server, and each output application

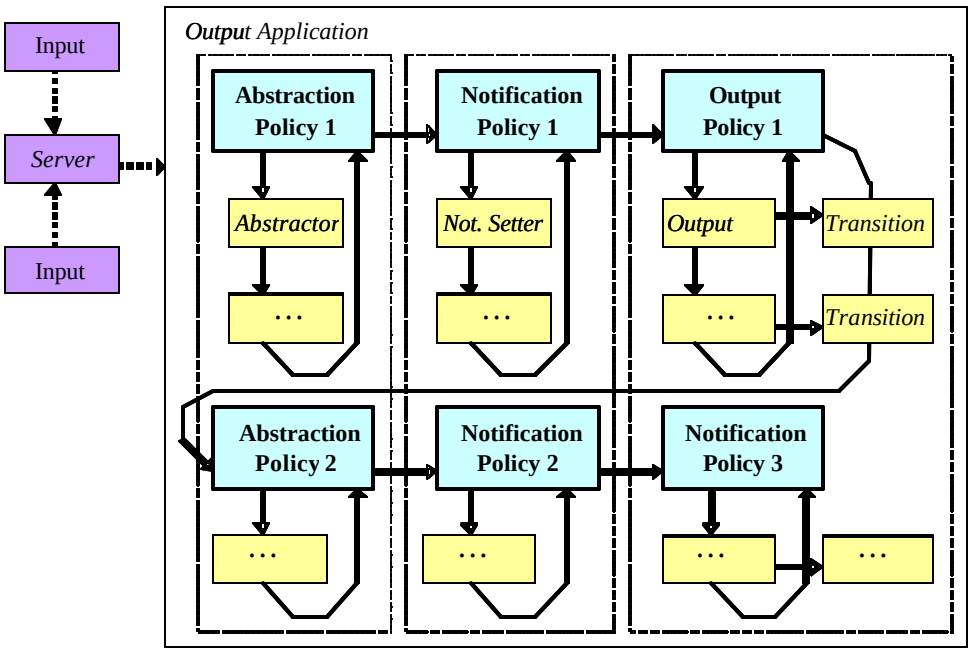

Figure 3
The PTK architecture. One or more inputs communicate with a server (far left). An application installs abstractors, notification setters, outputs, and transitions (italics) in their respective policies (bold). Input is then fed through each in turn.

maintains a runtime history of events for use by abstractors, notification setters, and transition classes.

To illustrate this more concretely, we next describe how data is represented in the toolkit, followed by the lifecycle of a particular piece of data, the movement of a bus in our bus display. We conclude with a brief description of the steps needed to create an application in the PTK. The following section describes separately how each of the three display characteristics, abstraction, notification, and transitions, are supported in the PTK.

Data and Templates

When data enters the toolkit, it is stored in basic data types: *binary* (such as on/off data), *number*, *number range*, *string*, or *file* (such as image, sound, or text data). Support for these data types was a result of our survey of existing displays: we did not find any display using data that would not fit into at least one of these categories. For example, the input source to the Bus LED provides the number of minutes left until a bus arrives.

One or more basic data types are stored in a *template*. Templates come in a variety of types, though untyped templates are supported. In addition to our basic data types, we include four template types in the PTK, though more could easily be added: audio, image, light, and turning motor. A typed template includes specific pieces of data and methods to manipulate that data. For example, the audio template includes the following data: sound (file of the audio recording), volume (a number range, with the minimum value being 0 and the maximum value being the maximum volume), and frequency (a number range).

Each template is identified by *metadata* objects. Metadata contains information about the input source generating the template. Metadata always includes the type of the template (audio, image, *etc.*) and it could also include location information and other descriptions. An application subscribes to receive events by telling the server what metadata it is interested in. For example the bus display subscribes to all templates of type "bus" with "bus number" matching any of six bus line numbers it displays.

The Lifecycle of an event

Each application receives data from the server and processes it in a main event loop. This loop sends the data to three policies, in order: abstraction policy, notification policy, and output policy (which includes transitions). Each policy passes the event to a series of abstractors, notification setters, and outputs, respectively. The policies implement rules for how the event is to be passed to these objects, in what order, and when control should be returned to the main event loop. As a result, an application developer can create complex effects by chaining together multiple abstractors or notification setters.

When the output application (the large box to the right of the server in Figure 3) receives an event from the server, the event is passed to the main event loop. Event handling and dispatch uses a similar architecture to the ArtKit and subArctic GUI toolkits [9,11]. The main event loop begins with a call to the `processEvent` method. This loops through lists of policies as depicted in Figure 3. We use a breadth-first traversal: each row of each policy list will execute together. For example, the k^{th} abstraction policy will pass the event to its abstractors, then the k^{th} notification policy will pass the event to its notification setters, and finally the k^{th} output policy will offer the event to its outputs. Once a row is complete, the main event loop will repeat this process for the next row, until all rows have been executed.

The toolkit provides default policies, abstractors, and notification setters. The defaults, derived from our survey,

are designed to be generic and are parameterized for the needs of specific applications. If the defaults are not appropriate, it is easy for a developer to create their own and install them. For example, it is easy to simply install a new notification setter in a notification policy. Similarly, an application developer can easily control which abstractors and notification setters are associated with each output device. If two outputs are installed in the same output policy, they will receive events that went through the same abstractors and notification setters. If in different policies, they will receive events that have been handled separately. Policies are rarely replaced, and none of our five test applications required any change to policies.

As an example of the event life cycle, events in the bus display begin with the bus input source, which reads bus schedules and determines how much time is left until six particular buses arrive. The input source wraps each number in a template, sets the metadata to indicate that it is bus arrival data and to specify which bus it represents, and dispatches it to the server. The server receives the six events and compares their metadata to the application's subscription's metadata. It sends the six events, one at a time, to the subscribing bus display output application.

The application's toolkit-provided communication class receives an event and sends it through the main event loop. First the event is abstracted to a light template by an abstractor from our library, installed by the application developer. The time until bus arrival is translated to the number of LEDs that should be turned on. Next the event is passed to the notification policy, which routes the event through two (library) threshold notification setters that look for different ranges of bus distance. With the notification set, the event is sent to the output policy, which loops through all six outputs (each represents a bus with a row of LEDs). The outputs check the metadata to see which bus the event represents and the appropriate output passes the event to its (custom) transition policy, after which the event is displayed.

Creating a Peripheral Display

Here we continue the bus display example. Its input object is a class that has information about each bus route, and generates events of type `Number` at one minute intervals indicating a bus' distance from the bus stop in minutes. There are six output objects, each a row of LEDs that can be individually controlled. The output objects make use of the Phidgets toolkit to control the LEDs [6]. The base classes that were extended to create these inputs and outputs handle communication with the PTK server automatically. All of this is similar to what one might create in other existing toolkits such as iStuff [2].

The bus display application encapsulates all six output objects. All are installed in the same output policy, and given copies of the same transition object. Thus, they share the same abstractors and notification setters: the single abstractor and two threshold notification setters mentioned above in the previous subsection. The application developer must also specify which input events are of interest, by creating metadata objects containing that information.

SUPPORT FOR THREE KEY CHARACTERISTICS

Each of the key characteristics of peripheral displays, abstraction, notification, and transitions, are supported by the PTK architecture and library. As previously described, each characteristic is important to peripheral displays. By providing toolkit support for them, our goal is two-fold. First, we hope to enable easier creation of peripheral displays. Second, we hope to encourage designers to think about these important issues when designing their peripheral displays. For example, toolkit level support for notification and transitions may help designers focus on design decisions affecting the kind of human attention their display is attracting.

Although our focus has been on the toolkit architecture, we have developed a rudimentary library of objects supporting abstraction, notification, and transitions. Additionally, we have developed five applications, discussed in more depth in the next section. Here, we describe the PTK's support for each key characteristic.

Abstraction

Abstractors convert between events of different types. We currently provide default abstractors that convert from input data to numbers, switches, audio, images, light, and motors (we plan to support other basic and template data types in the future). Application-specific feature abstraction is specified by overriding a `Translate` class and passing that in to the appropriate abstractor. For example, when converting a bus arrival time to a number of LEDs to light up (using the `ToLightAbstractor` abstractor), a developer provides a translator that translates the arrival time to an appropriate number range.

A custom abstractor may be implemented when a more complex analysis of sensed data is required. For example, we have written a recognizer that extracts telephone rings from non-speech audio. A developer can easily add other, more complex application-specific abstractors.

As stated above, the PTK architecture allows for multiple abstractors to be chained, meaning the results from one may become the input to another. For example, suppose we wanted to perform telephone recognition and then abstract the result to be displayed by a light. The output from the telephone abstractor would be the input to the `ToLightAbstractor`.

Notification

After an event is abstracted, its notification level is set. Notification levels are commonly chosen based on either a threshold, exact match, degree of change, or pattern match. All but pattern match are currently provided by the PTK library.

Thresholds: Thresholds are important when a peripheral display wants to set the notification level by the range in

which input data falls. In the bus display, notification is set to interrupt when a bus is within six minutes of the stop, but is set to make aware otherwise.

Exact match: An exact match requires data to exactly match a value provided by the application developer. For example, an email display might select the "interrupt" level when an email arrives from a specific author.

Degree of change: The degree of change notification setter compares the current event to the previous event, and determines the degree of change. A developer may specify different change thresholds for different notification levels. For example, a news ticker shows news headlines that change infrequently, so when a new headline appears, the user may wish to be made aware of it.

Pattern matching: Although not currently implemented, a notification setter might check for patterns in the data to select the notification level. For example, a display that visually shows the sounds occurring in a room may want to ignore background noise. If the pattern of the background noise can be determined, the notification level for this data could be set to ignore.

In addition, notification levels can be modified based on a local context sensor (we currently support ambient noise level sensing using a microphone). We can reduce notification levels if a noisy or busy environment is detected.

Transitions

Each output object has an associated transition class used to render the current notification setting. Transitions in the PTK are designed to allow for modular control over the exact behavior of the display as it transitions from the previous information event to the new event. A transition's primary role is to create a series of display events for the output object that provide a desired change of awareness for the user (such as change blind, make aware, or interrupt). These display events are displayed by the output object in the order they are generated. Transitions most commonly take the form of simple animations, and may have real-time constraints. The transition object spaces events to fit within a specified amount of time set by the application developer. The output class is then responsible for displaying information within those time constraints.

Our default transition class supports the major types of transitions found in our survey: smooth transitions (having many incremental display steps between the previous event and the new event), or abrupt transitions (show the new event without any intermediate transitional steps), or attention grabbing transitions (intermediate display steps are included that create sharp contrast to catch the user's focus). Thus, an event with an interrupt notification level generates a quick flashing sequence, while make aware generates an abrupt change, change-blind generates a smooth animation, and ignore is not displayed. By overriding this class, the toolkit user can arbitrarily define these fundamental animations or add any number of additional ones.

At a low level, this is supported as follows: The transition class generates default sequences of events for each type of notification (ignore, change blind, make aware, interrupt, and demand attention). To do this, it depends on two display-specific methods: `make_blank`, which should generate an "invisible" event, and `linear_map`, which should generate a linear interpolation between the old event and the new event, at intervals specified by the display designer. Additionally, it contains a method for each type of notification that implements the standard animations described above.

At a high level, the generic output class functions as follows. First it asks the transition class installed by the application developer to generate a sequence of events to be animated, given the previous and new event and notification levels. These events are placed in a queue. The output component then displays the events in order.

EXAMPLE APPLICATIONS

We have designed and implemented five applications using the PTK architecture: two physical output displays of bus schedule input; one on-screen output display of news and stock information; and two output displays of audio input. The applications were selected to demonstrate the different features of our toolkit. We re-implemented two applications developed as part of our past work in ambient displays (one bus [20], and one non-speech audio application [10]). Additionally, we re-implemented a third-party application (the news ticker) [21]. In each case, the flexibility of our architecture allowed us to expand on the features supported in the past. Together, these applications demonstrate that the PTK enables easier support for the key features of peripheral displays, supports easier prototyping, and that it can allow applications to be defined in terms of issues relating to human attention such as notification and transitions.

Bus Mobile

The Bus Mobile, shown in Figure 4 (left), gives users a sense of how much time is left until popular buses reach their chosen bus stops [20]. It includes six tokens, each representing a bus line. At 24 minutes before a bus arrives at its stop, its token lowers 24 inches below the mobile's skirt. It rises one inch every minute until it disappears under the skirt when the bus has left the bus stop.

A heuristic evaluation of the original Bus Mobile found several usability problems. First, the Bus Mobile did not properly use notification. The major notification happened at 24 minutes when the bus token lowered from 0 to 24 inches. This action interrupted users rather than making them aware of the approaching bus. The most important notification needed was when the user actually had to leave for the bus stop, when the token was about 5 inches from the top. This event was not distinguished in any way from other events, making it essentially change blind to users.

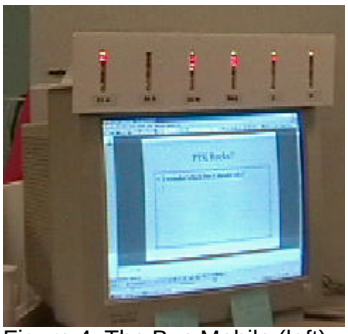

Figure 4: The Bus Mobile (left) and Bus LED (right, also shown in Figure 1)

Use of PTK architectural features

We re-implemented the Bus Mobile with the PTK. Since the Bus Mobile did not make good use of notification or transitions, we did not find any savings in the reimplementation as far as lines of code or code complexity. The input and output code in the original Bus Mobile were highly integrated, causing the two pieces to be indistinguishable. Using the PTK separated the two, allowing us to reuse the bus schedule input and much of the output application in the Bus LED Display.

Bus LED

The Bus LED Display, shown in Figure 4 (right), took into consideration the usability problems of the Bus Mobile, assigning more appropriate notifications and transitions to events. The display consists of six columns of LED lights, arranged horizontally, labeled below with the corresponding bus number. There are eight LED lights in each column whose on/off values are controlled by a Phidget interface kit [6]. Each LED corresponds to a three-minute window. The LED lights turn on from top to bottom; *e.g.* when a bus is eight minutes away, the top six LED lights are on. When the bus is six minutes away, the LED lights flash on and off a few times to catch the user's attention (an "interrupt" event). Other than the one flashing event, the LED light changes are as unobtrusive as possible to keep the display in the user's periphery.

Use of PTK architectural features

The PTK implementation of the Bus LED Display used the same input as the Bus Mobile and required only small modifications to the Bus Mobile code. Changes included using a different abstractor to translate minutes to number of LED lights and changes to the parameters for the notification setters. Notification was set to interrupt when the bus was six minutes away, make aware when the bus was between 1 and 24 minutes away, and ignore otherwise. The output class was re-implemented, since new hardware was used. The transitions class was modified by overriding `make_blank` to turn all of the lights off, with the result that an interrupt transition would flash the lights on and off.

Stock-News Displays

As a demonstration of our support for a variety of transitions and for alerting displays, we chose to re-implement a modified version of the information ticker

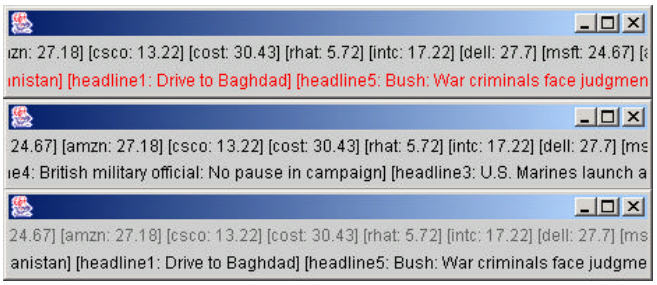

Figure 5: Three images of the stock and news ticker. **(top and middle images)** The news headline flashing red to black **(bottom image)** The stocks fading in.

presented by McCrickard [21]. In addition to displaying news (we chose to show the top five headlines from CNN.com), our ticker displays current stock prices. Figure 5 shows images of the ticker, with stocks on top and news below.

McCrickard's original ticker was a simple scrolling-text display. The text was only updated when it was not visible. Hence, all information updates were change-blind. Our stock and news ticker supports arbitrary notification levels and transitions. Updates are change blind by default. When Intel's stock changes by a small amount, an update is set to make aware. When the word "Iraq" appears in a headline, or a very large change in Intel's stock occurs, an update is set to interrupt. Change-blind transitions were implemented as fade-out/fade-in transitions where information updates were conducted while the text was transparent. Make-aware transitions were implemented as a single flash, during which the scrolling text was turned green. Interrupt transitions were implemented as multiple flashes, during which the text color altered between red and its usual black.

Use of PTK architectural features

The stock and news ticker require a web page parser, which is part of the toolkit library. No customized notification setters were required. To support the special transitions described above, four methods had to be subclassed to perform display-specific actions: the standard make aware method was modified to flash the text in green once, the standard interrupt method was modified to flash from black to red instead of black to blank, `make_blank` was modified to generate transparent text, and `linear_map` was overridden to fade between events.

Remote audio awareness -- Ring Ticker

The Ring Ticker (top of Figure 6) is designed for those who cannot easily hear important audio events. Its design is based in part on our recent work in peripheral sound displays for the deaf [10]. Currently, the ticker provides awareness of one type of sound: telephone ring tones. When the phone rings, the word "ring!" slowly fades into view as it scrolls across the ticker.

Use of PTK architectural features

This display demonstrates feature abstraction (our abstractor "recognizes" rings based on key frequency features present across events). This is the most complex

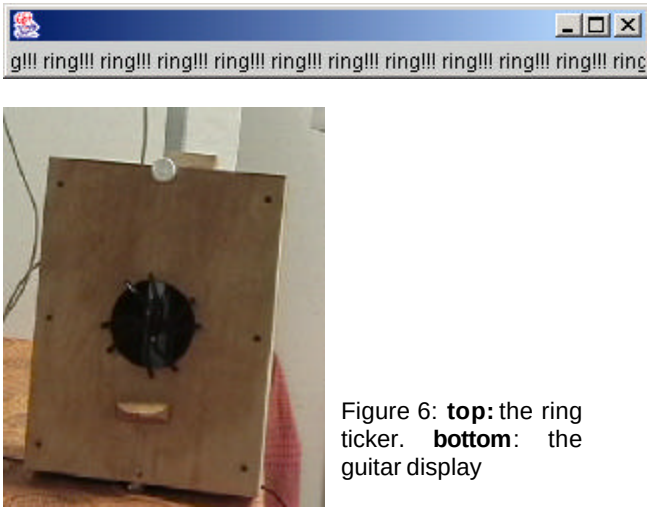

Figure 6: **top:** the ring ticker. **bottom**: the guitar display

abstractor we built. It also demonstrates re-use: it was built using the same output class and transition class as the stock-news display. We simply installed a different abstractor (the ring recognizer), and different notification level setters. Notification level is always set to either "make aware" (which causes the word "ring!" to appear) or "ignore" (which causes it to disappear again).

Remote audio awareness -- Guitar

The Guitar, shown at the bottom of Figure 6, provides awareness of audio events occurring in a remote space by plucking a string. A single vertical string runs down the center of the instrument. A Phidget servomotor in the black central area pulls a guitar pick across the string, whose tension is controlled by another servomotor not visible in the image. Thus, the display can control pitch and the frequency with which the string is plucked. The result is an audible indication of activity levels in a remote space. The greater the activity, the higher the pitch and more frequent the plucks.

Use of PTK architectural features

Both the Guitar and the Ring Ticker used the same input source simultaneously. The Guitar used a different abstractor and notification setters. It used an abstractor to convert remote audio volumes to servomotor positions. Notification levels were set to change blind, make aware, or interrupt, using a threshold setter, based on volume.

Discussion

Our applications demonstrate that the PTK facilitates code reuse, enables easy interface experimentation, and supports the design of displays that consider human attention through notification and transitions.

The two bus arrival displays demonstrate the advantages of our architecture in supporting a design that is focused first and foremost on issues of attention. The ticker demonstrated our ability to implement an example application from the literature, and highlighted the flexibility of our transition system. Finally, the two audio displays demonstrated the ability of the PTK to take the same input and support applications with varying levels of abstraction. Two displays can show different characteristics of the audio input: volume, frequency, or both.

Among the five applications we have built, two did not use abstraction, we were able to use the default classes with custom translators in three cases, and we created one complex abstractor, the ring recognizer.

Our default notification setters can be parameterized to meet most application needs. For example, among the five applications we have built, we were able to use the default classes in four applications, and created two subclasses (most applications used multiple notification setters). It is easy to create custom notification setters when necessary.

In the three applications we implemented that used transitions, we relied on the standard transition functionality, overriding methods only to add display-specific functionality. Two of the applications are GUI displays while the third displays transitions on LEDs. Transitions could also be done in audio or other modalities.

Together, we feel that these applications illustrate the flexibility and strength of the characteristics we support in the toolkit.

CONCLUSIONS AND FUTURE WORK

Based on a survey of existing peripheral displays and cognitive science literature, we have characterized peripheral displays according to their use of three key characteristics: abstraction, notification, and transitions.

We then developed a tool to support the building of peripheral displays, based on these characteristics. In addition to supporting basic issues highlighted by other toolkits in this domain, including distribution and discovery of available input sources, our tool provides explicit support for each of the three characteristics we identified.

We have validated this tool by building a total of five peripheral displays, representing variations on three different applications. Two are previously-built applications that we re-implemented with the PTK [10,20], and a third is an application originally presented in the literature [21]. Each of these makes use of a variety of library and custom abstraction, notification, and transition objects.

In sum, these five displays illustrate the importance of the characteristics highlighted in our survey, and show that by supporting them, the toolkit eases prototyping, enables interface experimentation and supports the design of displays that consider human attention through notification and transitions.

As always, this is a work in process. We have focused little effort on creating a truly complete toolkit library. In the future, we plan to conduct studies that can help us to determine how best to populate our library. In particular, we will collaborate with cognitive scientists in determining the amount of attention different transitions require of users. Based on these studies, we will expand the range of

animations supported by our transition implementation, and include more sophisticated support for animation in our transition class [12]. We also plan to include a larger variety of data types, including streaming data.

We are also interested in expanding the interpretation of local context currently available to transition classes in the toolkit. For example, we hope to leverage off the work of Hudson *et al*. [13], who explored different sensors appropriate for determining interruptibility.

Finally, we plan to use our toolkit to continue to build and evaluate a variety of peripheral displays, expanding our understanding of the evaluation techniques and tools necessary to develop these intriguing computational devices, and in the process learn about the factors that influence the success of these displays.

REFERENCES

1. Abowd, G.D., *et al.*, "The Human Experience." IEEE Pervasive Computing **1**(1):48-57. 2002.
2. Ballagas, R., *et al.* "iStuff: A physical user interface toolkit for ubiquitous computing environments". In *Proc. of CHI 2003*. To appear.
3. Cadiz, J. *et al.* "Designing and deploying an information awareness interface". In *Proc. of CSCW'*02, pp. 314-323. 2002.
4. Chang, A., *et al*. "Lumitouch: An emotional communication device". In *Extended Abstracts of CHI '01*, pp. 371-2. 2001.
5. Dahley, A. *et al.* "Water lamp and Pinwheels: Ambient projection of digital information into architectural space." In *Extended Abstracts of CHI'98*, pp. 269-270. 1998.
6. Fernandez-Duque, D. and Thornton, I.M. "Change detection without awareness: Do explicit reports underestimate the representation of change in the visual system." *Visual Cognition*, **7**: 324-344.
7. Greenberg, S. and C. Fitchett. "Phidgets: Easy development of physical interfaces through physical widgets." In *Proc. of UIST '01*, pp. 209-218. 2001.
8. Heiner, J. M. *et al.* "The Information Percolator: ambient information display in a decorative object", In *Proc. of UIST '99.* pp. 141-148. 1999.
9. Henry, T.R., *et al.* "Integrating gesture and snapping into a user interface toolkit." In *Proc. of UIST '90*, pp. 112-122. 1990.
10. Ho-Ching, W. *et al*. "Can you see what I hear? The design and evaluation of a peripheral sound display for the deaf." In *Proc. of CHI'03.* To Appear.
11. Hudson, S.E. and Smith, I. "Supporting dynamic downloadable appearances in an extensible user interface toolkit." In *Proc. of UIST'97*, pp. 159-168. 1997.
12. Hudson, S.E. and Stasko, J.T. "Animation support in a user interface toolkit." In *Proc of UIST'93*, pp. 57-67. 1993.
13. Hudson, S.E. *et al*. "Predicting human interruptibility with sensors: A wizard of oz feasibility study." In *Proc. Of CHI'03*. To appear.
14. Intille, S. S. "Change blind information display for ubiquitous computing environments." In *Proc. of Ubicomp '*02, pp. 91-106. 2002.
15. Linnett, C. *Perception Without Attention: Redefining Preattentive Processing.* PhD Thesis. UC Berkeley. 1996.
16. MacIntyre, B., *et al.* "Support for multitasking and background awareness using interactive peripheral displays. In *Proc. of UIST '01*, pp. 41-50. 2001
17. Mack, A. and Rock, I. *Inattentional Blindness*. MIT Press, Cambridge. 1998.
18. Mack, A. "Perceptual organization and attention." *Cognitive Psychology*. **24**:475-501. 1992.
19. Maglio, P.P. & Campbell, C.S. "Tradeoffs in displaying peripheral information." In *Proc. of CHI'00*, pp. 241-248. 2000.
20. Mankoff, J. *et al.* "Heuristic evaluation of ambient displays." In *Proc. of CHI'03*. To Appear.
21. McCrickard, D. S. and Zhao, Q. A. "Supporting information awareness using animated widgets." In *USENIX Technical Program*, pp. 117-127. 2000.
22. McCrickard, D. S. *et al.* "Supporting the construction of real world interfaces." In *Proc. Of HCC'02*, pp. 54-56. 2002.
23. Miller , T. and Stasko, J. "InfoCanvas: A highly personalized, elegant awareness display", in *Supporting Elegant Peripheral Awareness, workshop at CHI '03.* To Appear. 2003.
24. Myers, B. *et al.* "Past, present and future of user interface software tools." ACM TOCHI, **7**(1):3-28. 2000.
25. Mynatt, E.D., *et al*. "Digital family portraits: Providing peace of mind for extended family members." In *Proc. of CHI'01*, pp. 333-340. 2001.
26. Mynatt, E.D., *et al.*"Designing audio aura". In *Proc. of CHI '98*, pp. 566-573. April 1998.
27. Pedersen, E. R., and Sokoler, T. "AROMA: Abstract representation of presence supporting mutual awareness." In *Proc. of CHI'97*, pp.51-58, 1997.
28. Posner, M.I, and Petersen, S.E. "The attention system of the human brain." *Review of Neuroscience*, **13**: 25-42. 1990.
29. Redström, J. *et al.* "Informative art: using amplified artworks as information displays." In *Proc. of DARE'00.* pp.103-114. 2000.
30. Rensink, R. "Change detection." *Annual Review of Psychology*, **53**:245-77. 2002.
31. Tran, Q., and Mynatt, E. D. "Cook's collage: Two exploratory designs." In *Technologies for Families, workshop at CHI'02*. 2002.
32. Treisman, A. "Distributed attention." In *Attention: Selection Awareness and Control*. Clarendon Press, Oxford. pp. 5-35. 1993.
33. Weiser, M. and Brown, J.S. "Designing calm technology," *PowerGrid Journal*, **1**(1), July 1996.
34. Weiser, M. "The computer for the 21st century." *Scientific Americ*an, **265**(3):94-104. 1991.