A Toolkit for Managing User Attention in Peripheral Displays

Tara Matthews, Anind K. Dey[*], *Jennifer Mankoff, Scott Carter and Tye Rattenbury*
EECS Department, UC Berkeley
Berkeley, CA 94720, USA
[*]Intel-Berkeley Research, 2150 Shattuck Ave, #1300
Berkeley, CA 94704, USA

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* can allow a person to be aware of information while she is attending to some other primary task or activity. We present the Peripheral Displays Toolkit (PTK), a toolkit that provides structured support for managing user attention in the development of peripheral displays. Our goal is to enable designers to explore different approaches to managing user attention. The PTK supports three issues specific to conveying information on the periphery of human attention. These issues are *abstraction* of raw input, rules for assigning *notification levels* to input, and *transitions* for updating a display when input arrives. Our contribution is the investigation of issues specific to attention in peripheral display design and a toolkit that encapsulates support for these issues. We describe our toolkit architecture and present five sample peripheral displays demonstrating our toolkit's capabilities.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Design Tools and Techniques — *software* libraries; user interfaces. H.5.1 [**Information Interfaces**]: Multimedia Information Systems —*artificial, augmented, and virtual realities*. H.5.2 [**Information Interfaces**]: User Interfaces — *input devices and strategies; interaction styles; prototyping; user-centered design*.

Additional Keywords and Phrases: Peripheral displays, user attention, toolkits

INTRODUCTION
Everyday environments include many sources of information that we monitor almost without realizing it. Windows let us know the weather, the approximate time of day, and the level of activity nearby. Clocks allow us to monitor time, and notify us with alarms when we need to change activities. Footprints or tracks give us a sense of who or what has passed through an area and how long ago.

UIST '04, October 24–27, 2004, Santa Fe, New Mexico, USA.

At any given time, most of the information in our environment is *peripheral* to our main focus of attention. Similarly, in a ubiquitous computing environment that involves more than one device, most devices and displays must be peripheral. *Peripheral displays*, then, are an important class of ubiquitous computing applications that can allow a person to be aware of information without being overburdened by it [30]. Peripheral displays generally require minimal attention and cognitive effort, and are not part of a user's primary activity.[1]

As an example of a physical peripheral display, consider the "Motion Monitor" shown in Figure 1. There is a venerable history of using peripheral displays to show activity, including Ishii's pinwheels [14] that show activity via pinwheel motion, and the dangling string [30] that spins with a speed indicating network activity levels. The Motion Monitor continues in this tradition by indicating how much activity is occurring in a remote location, allowing a user to maintain a sense of connectedness with a remote colleague or friend despite geographic distance. The activity is sensed by a camera and analyzed using image differencing. It is then displayed remotely on a commercial Ambient Orb™ (a glass orb that changes colors). The orb color changes more rapidly when activity in the remote location increases. It allows a user to monitor remote activity while focusing on a separate, primary task, like reading.

Although much effort has focused on understanding and designing peripheral displays in recent years [7,9,14, 21,22,24], there is general agreement in the research community that they are difficult to design, build, and evaluate [9,18]. In interviews of ten peripheral display creators, we found that difficulty prototyping these displays has led to fewer iterations


Figure 1: Motion Monitor

[1] Research in this area often refers to both ambient and notification displays. Typically these terms imply certain approaches to conveying information to the user. Notification displays tend to draw focal attention at key moments, while ambient displays typically avoid drawing focal attention and instead use divided attention. See the next section for a discussion of focal and divided attention and their treatment in the PTK.

in practice. Although toolkits exist that may aid building peripheral displays in a mechanical sense [1,4,8,21,24], they do not address attention issues, a crucial problem for peripheral display designers. We present the PTK, a toolkit to support the creation of peripheral displays, focusing specifically on managing user attention.

Peripheral displays must address two sets of attention issues. One, an active area of current research, deals with context about the user, including her interruptibility [6,11], current primary activity or task [4,11], and focus of attention [11]. Though addressing this first attention issue is not our focus, our toolkit does allow designers to make use of existing research in sensing user context. The other attention issue for peripheral displays, and the focus of this paper, deals with *attention management,* or how a display can convey information appropriately. We chose to focus on this issue for two reasons. First, user context is difficult and invasive to sense. It is not practical to assume all peripheral displays will have access to it. Second, our toolkit is structured to focus on *management* issues such as abstraction and transitions, which are independent of user context. Our goal is to allow a designer to explore questions such as: "Does a display interrupt the user?" "Is a change noticeable the next time she glances at a display?" "When should a display address the user?" We present an analysis of and toolkit-level support for managing user attention through peripheral display design, which we will refer to as *managing user attention* throughout this paper. Note that our contribution in this paper is *not* to create better displays, but to *empower* a display designer to make attention management explicit in the design of his display.

The Peripheral Displays Toolkit (PTK) allows a designer to assign *notification levels* to information, indicate how a display should *transition* from showing one notification level to another, and *abstract* information so that it is easy to understand by glancing at it. Abstraction, notification levels, and transitions are important because they allow designers to manage how peripheral displays impact attention. In our analysis of existing peripheral displays, these three issues arise repeatedly.

Our contribution lies in the architecture and library that we designed to support the key issues of abstraction, notification levels, and transitions. A secondary contribution is the identification of the three issues, which is based on an inspection of existing peripheral displays and cognitive science research on human attention.

Overview

The next section presents the three characteristics supported by our toolkit (abstraction, notification levels, and transitions), justifying them based on existing peripheral displays and literature on attention. We then present our architecture and library, describing how we support each of these characteristics. 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.

MANAGING ATTENTION WITH ABSTRACTION, NOTIFICATION LEVELS, AND TRANSITIONS

Peripheral displays are commonly used as *secondary* sources of information, separate from a user's primary, focal task. The information they display varies in its importance to the user. Thus, the method of display should vary and should require varying amounts of user attention. For example, a clock (usually low importance) is typically monitored by a user at her convenience, while an alarm clock may make a sound that is difficult to ignore at an important moment. We believe that a display must support three key characteristics in order to appropriately manage the connection between information importance and user attention: **abstraction, notification levels** and **transitions.** First, a display must determine how to show information. A display is easier to monitor when it can be read at a glance. To allow easy monitoring, the information being displayed may need to be *abstracted.* For example, our Motion Monitor replaces a raw video stream with color indicating activity. Second, a display must specify the importance of making the user aware of a piece of information, by selecting a *notification level*. The Motion Monitor maps different amounts of activity to different notification levels. Lastly, a display must change to indicate the arrival and importance of new information. We refer to this change as a *transition.* For example, the Motion Monitor changes color to indicate new information, using varying intensity and speed, based on the current notification level, to indicate the importance of the information being displayed.

What follows is a discussion of these three characteristics, based on a literature survey of 15 existing displays (described in [19]), cognitive science literature, and our own experience creating and evaluating displays.

Abstraction

Abstraction involves extracting features or reducing the fidelity of information so that it is easier to read "at a glance" than the raw input. Abstraction enables lower-attention consumption of information. One reason for this may be that abstract (pictorial) displays aid recall when compared to literal (textual) displays [25,28]. The AROMA project [24] showed that abstracted data can convey sufficient information while remaining subdued enough to allow a user to concentrate on a main activity.

The Motion Monitor uses image differencing to convert the incoming camera data to events representing activity level. While most of the existing displays we surveyed also abstract data in some way, the practice is particularly common among ambient displays. For example, AROMA [24] abstracts video and audio into an activity level and displays it using sound, heat, motion and graphical elements. Audio Aura [23] abstracts the time since a group member was last seen into presence information, displayed via audio cues. Lumitouch [2] abstracts motion in front of a picture frame to presence information, displayed with LEDs on a paired picture frame in a remote location.

Notification Levels

We use the term *notification levels* to refer to differences in information importance. Higher notification levels correspond to more critical data and are displayed in a way that grabs a user's attention. Lower notification levels correspond to non-critical data and are typically displayed in a way that does not attract attention, but allows a user to monitor a display occasionally or peripherally. For example, the LEDs in Lumitouch glow when someone in a remote location is present and flash when she squeezes it hard [2]. Thus, presence is assigned a different level of importance than interaction with the display. The choice of how to *display* information at a given notification level is a separate issue that we address next, in our discussion of transitions. Here we focus on the question of how to *assign* a notification level. Though notification levels are a key characteristic of peripheral displays, the basis for their choice is rarely made explicit. We thus ground our discussion of notification levels in attention literature.

Based on recently developed models [14,16,26,27,29], we categorize human attention (*i.e.,* intent and expectation towards a stimulus [17]) into three types: *inattention, divided attention,* and *focused attention*. In *inattention*, objects are not directly available for conscious awareness but may still affect behavior (*e.g.*, memory recall) [5]. *Divided attention* and *focused attention* represent the two ways that humans consciously perceive stimuli – by distributing attention over several objects or using all attentional resources to focus on one stimulus, respectively.

Notification levels represent relative information importance. The more important a stimulus, the more attention it should consume[2]. Thus, an important event might demand focused attention, while a less important event might be monitored with divided attention. We define five notification levels, based on the types of attention just discussed: *ignore, change blind, make aware, interrupt,* and *demand action*. *Change blind* represents information of marginal importance that should be displayed in a way that consumes no conscious awareness but may still affect behavior, corresponding to inattention [13]. *Make aware* represents information of some importance that should consume some attention, corresponding to divided attention. These two levels are typically used by ambient displays such as an Ambient Orb™. *Interrupt* and *demand action* represent information that should grab focused attention, temporarily distracting the user from her primary activity. These levels are typically used by notification displays such as an alarm. *Demand action* also requires that the user perform some action to stop the alerting. These levels, while common in Graphical User Interfaces (GUIs), should be used sparingly in peripheral displays, because only critical information should require the user to drop everything and attend to the displayed information. *Ignore* represents information that is unimportant and should not be displayed, not corresponding to any attention type. Thus our notification levels span a range from information of least importance that should demand no attention to information that should demand full attention.

While it may seem contradictory for a *peripheral* display to attract *focused* attention, this enables a display to be peripheral when it is not displaying important information. A watch alarm only requires focused attention when it goes off, alerting the user that it is time to go somewhere. If a user could not count on the alarm to attract her full attention, she would pay more attention to the watch.

The existing displays we surveyed used all but the *demand action* notification level. *Make aware* was by far most common, especially among ambient displays, though several displays used *change blind* and notification displays tended to use *interrupt*. Our Motion Monitor is a typical example of this. It assigns different notification levels to different amounts of activity based on the activity level returned by its abstractor. Low and high levels of activity are mapped to the change blind and make aware notification levels, respectively. Because high remote activity is not an overly important event, the Motion Monitor does not use the *interrupt* notification level.

Transitions

It is difficult to describe how a display employs notification levels without describing the *transitions* it employs. Transitions are a mechanism for creating effects on a display that attract an appropriate amount of attention from the user, based on a new event's notification level. The Motion Monitor changes colors with different speeds depending on the notification level of incoming events, for example. These animated color changes are transitions.

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. Displays attempting to attract focal attention for important information (*interrupt* level) typically utilize abrupt transitions. Several studies [2,7,23] have shown that significant changes in the interface will draw a user's attention. Displays attempting to capitalize on divided attention (*make aware* level) employ a number of more subtle techniques, such as updating small pieces of the display abruptly [22], or including slow movements [18]. For displays utilizing inattention (*change blind* level), McCrickard *et al.* found that animations like fading, rolling, and tickering were not distracting to low-attention primary tasks [20]. This suggests that repetitive and very gradual animations are appropriate for change blind transitions in this context. Further research is needed to determine how to best utilize inattention in other contexts.

Based on these results, it is clear that animations of different types are a key mechanism for supporting transitions in applications that do not want to distract users.

---

[2] Though we focus on data importance independent of context, a designer can incorporate context about a user into decisions about notification levels by writing a custom notification map using external context sensing solutions such as [11,6].

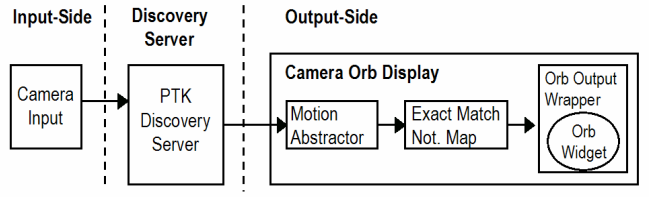

Figure 2: Motion Monitor architecture diagram

Our survey confirmed that displays explicitly minimizing motion are more likely to be change-blind [13]. Other systems, such as InfoCanvas [22], may make the user aware of new information by changing or moving an image more quickly or using other forms of animation. We also found that some displays used abrupt or significant motion to purposefully interrupt [2].

Managing Attention Summary

To summarize, peripheral displays convey information of varying importance, and it is crucial that they do this in a way that appropriately manages their impact on a user's attention. We have identified three important characteristics of peripheral displays that can be manipulated by designers to control the attention required of users. While the design space defined by these three characteristics is large it is also well populated by existing displays [19]. By explicitly addressing all three, the PTK supports current practice, and helps designers to focus on managing user attention in their displays. This supports the design of displays that more carefully consider human attention, as shown by our example applications later in the paper. Next we discuss the PTK architecture and its support for these characteristics.

PTK ARCHITECTURE

The PTK provides three main areas of support for peripheral display development: (1) architectural support for managing impact on human attention using techniques described in the preceding section; (2) simplified display design and code re-use; and (3) a library of common peripheral display components that can be pieced together as is or used with custom application code. In this section, we focus on our support for managing human attention through abstraction, notification levels, and transitions and we demonstrate how these concepts are leveraged to quickly build peripheral displays.

Creating a Peripheral Display

A peripheral display is composed of some number of the following components: input sources, output widgets, abstractors, and notification maps. These components are *connected* to each other in a `PeripheralDisplay` class. Recall the Motion Monitor (Figure 1) that uses a camera to detect remote presence information and displays it on an Ambient Orb™. Figure 2 shows this display's architecture.

Using this display as an example, the `PeripheralDisplay` class code for connecting the application components is as follows:

```
1  cam_in = new CameraInput();
2  orb_out =
     new MotionMonOutput(new MotionAbstractor());
3  match = new ExactMatchNotificationMap(
     AMOUNT_OF_MOTION, no_motion, lo_motion,
     hi_motion, null, null);
4  makeConnection(cam_in, orb_out, match);
```

The camera input, orb output widget wrapper, motion abstractor, and a notification map are instantiated in lines 1 - 3. In line 4, these components are connected using `makeConnection`, which sets up the event flow from input to output. First, the input receives an image from a camera and sends it in an event to the motion abstractor, which uses image differencing to determine how much motion has occurred since the last event. Motion is specified by the percentage of pixels that have changed between consecutive image frames. The abstractor adds this numeric data and a customizable semantic ranking of motion (none, low, or high) to the image event. The PTK then sends the abstracted event to the notification map, which looks for a match between its parameters and the semantic ranking data, and sets an appropriate notification level for the event. In our example, we map the event to *ignore*, *change blind*, or *make aware* if its `AMOUNT_OF_MOTION` data matches `no_motion`, `lo_motion`, or `hi_motion`, respectively. (The *interrupt and demand action* parameters are `null` indicating they are not being used in this display.) Finally, the PTK sends the event to the orb output widget wrapper, which displays the event using different transitions based upon its notification level.

The `MotionMonOutput` output widget wrapper extends a generic `OrbOutput` widget with application-specific code that indicates how to display the remote presence event using transitions (*i.e.*, which `OrbOutput` widget methods to call and the parameters to pass them). In general, when using PTK library output widgets, a wrapper is required to display events in an application-specific manner.

With the input, output, abstractor, and notification map above all being PTK library elements, this (along with 9 lines of output wrapper code) is all that is needed to create this display. There are two important details to note. First, this code example is for an application where all the components are being executed on a single machine. The PTK allows applications to be distributed across machines in different locations with minimal code changes. Second, the Motion Monitor uses a simple event flow with a single instance of each type of component. In the next section, we describe how an application may combine multiple inputs, outputs, abstractors and notification maps by chaining or aggregating them.

Combining Multiple PTK Components

An application developer may wish to create a display that uses multiple inputs or outputs. For example, on the input side, the Motion Monitor might also use microphone input along with a camera to sense activity. Alternatively, InfoCanvas [22], a system that maps different input sources to animated, on-screen images, could be implemented in the PTK by using a variety of input sources combined with a separate output widget to represent each input source as an

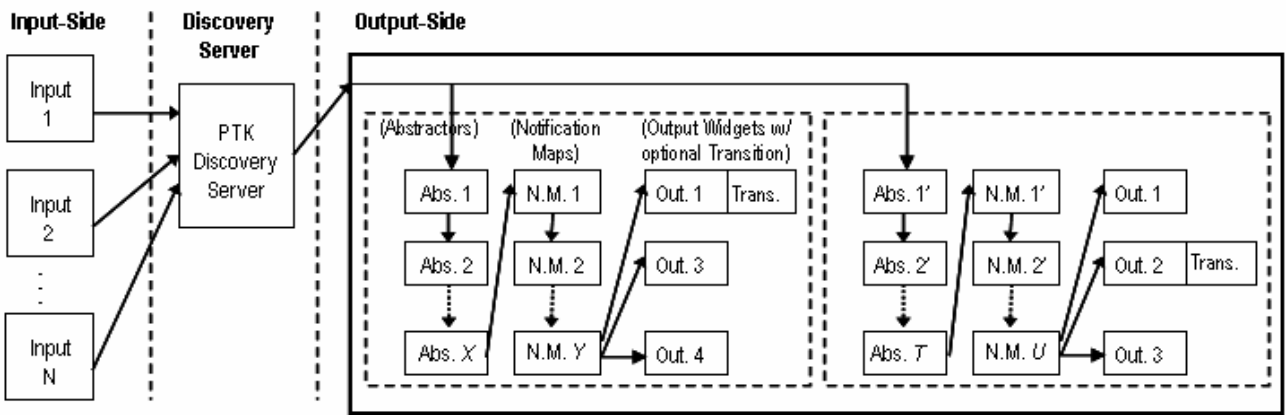

Figure 3: Architecture diagram of a general display. Inputs are connected to collections of abstractors, notification maps and output widgets. Each collection may contain different components, or some components may be used in multiple collections

animated image. The PTK can support both of these examples through aggregation of multiple input sources or output widgets (illustrated in Figure 3).

Aggregation is handled by passing arrays of multiple inputs and/or output widgets to `makeConnection`(). By default, this causes events from all inputs to be delivered to all output widgets. However, if desired, a developer can specify that certain inputs should only go to certain outputs (*via* multiple calls to `makeConnection()`).

Additionally, to create more sophisticated abstractions, individual abstractors can be chained together in a list, such that the output of one becomes the input of another. For example, camera input could be passed through three abstractors: motion, people counting, and *activity*, which uses motion and number of people to assign a low, medium, or high activity level. Similarly, notification maps may be chained together. Examples of abstractor and notification map chains are shown in Figure 3.

Addressing Human Attention

In the previous subsections, we demonstrated how one uses components of the PTK to build peripheral displays. Here, we describe toolkit support for abstractions, notification levels, and transitions, that provides structure to help developers address issues related to human attention.

Abstraction

After an input creates an event, the event is abstracted. Abstraction transforms input data to a form that is *semantically compatible* with questions a user might care to ask about the data. For example: Is the person in the room? How many people are in the room? Is the phone ringing? To be answered, these questions require abstraction (*e.g.,* recognizing phone ring tones given audio input data).

It is possible to perform abstraction on either the input or the output side. Typically, abstraction is done on the output side. Output side abstraction is useful because it allows non-abstracted input to be used by multiple applications at once. There are cases in which input side abstraction is appropriate: when an input produces large or frequent event data (*e.g.,* images, audio), or the event data is private in nature (abstraction can extract only the relevant details and degrade or remove private data). A developer specifies where abstraction should occur by passing an abstractor (or a list of abstractors, for chaining) to the appropriate class (`Input` or `Output`).

Notification Levels

After an event is abstracted, its notification level is selected. The PTK library includes *notification map* components for setting notification levels. Notification levels are set after abstraction is completed because they deal with data as it will be presented to an output widget. In our literature survey [19], we found that notification levels are commonly chosen by determining if the incoming event's data falls within some range (*threshold*), is equal to or contains a specified value (*exact match* or *contains*, respectively), or has changed by a certain amount (*degree of change*). The PTK library includes all of these.

As an example of a *threshold*, in a remote presence audio display, if the volume of an audio event is above a certain level, or threshold, a high notification level may be chosen. The Motion Monitor checks for *exact matches* to the semantic motion values produced by the motion abstractor (none, low, high) to determine the notification level. Using *contains*, a news ticker may interrupt when a key word appears in any news headlines. Checking for a *degree of change*, a stock ticker may interrupt when a stock's value changes by more than five points.

Though the PTK does not directly integrate information about the user's current context into its architecture, a designer could incorporate information from a third party system that senses user context into a notification map, such as work by Horvitz and Apacible [11] or Fogarty *et al.* [6]. Such a map could determine the notification level of an event by combining the data's importance and the interruptibility of the user, or other user state.

Transitions

When an output widget receives an event with a notification level, it uses a transition to render the event. Transitions are a mechanism for displaying new incoming

events on a specific output widget. The PTK provides general architectural support for transitions by asking the developer to implement certain key methods.

To include transitions, an output widget class implements the `Animatable` interface. In simple cases that do not require animation, the application developer may choose to bypass transitions by not implementing `Animatable`. The architecture uses the methods defined in the `Animatable` interface to create a queue of events that represent an animation. Those events are then passed, in order, to the output widget to be displayed via calls to `displayEvent()`. The application developer can control the time between events, the number of events in the queue, and the way events in the queue are displayed.

The architecture creates the queue by calling the following series of methods from the Animatable interface: `startTransition(),[transitionStep(stepnum)|alternateStep(stepnum)]…[transitionStep(stepnum)|alternateStep(stepnum)],endTransition()`. `startTransition()` and `endTransition()` return events specifying the initial and final state of the output widget, respectively for this transition. `transitionStep(stepnum)` returns an event specifying the state of the output widget at a particular step during the transition. `alternateStep(stepnum)` should return the inverse of the event that `transitionStep()` returns. The number of steps, and the use of `transitionStep()` and `alternateStep()` varies depending on notification level. For example, an interrupt transition could use `transitionStep` and `alternateStep` repeatedly to create a flashing effect. The default animations associated with each notification level are:

- **ignore**: returns nothing; event not displayed.
- **change blind**: creates a list of events using `transitionStep()`, creating a gradual change.
- **make aware**: returns a single event by default, creating a more abrupt change, with the option to create a list of events for a more gradual change.
- **interrupt**: creates a list of events using `transitionStep()` and `alternateStep()` one after another, creating a flashing effect.
- **demand action**: creates a series of `interruptTrans()` animations, repeating until the user performs an action specified by the developer.

Animations may be customized for a given notification level. They may also be assigned based on the notification level of the current and previous events. A transition from one notification level to another may need to be unique (*e.g.,* an "interrupt to change blind" transition might cause a different output action than "make aware to change blind"). The PTK provides an interface for customizing all permutations of one notification level to another (`interrupt2changeblind`, `interrupt2makeaware`, *etc*.).

Support for transitions is architectural, so there are no library components. The PTK includes three reusable examples of `Animatable` output widgets: one for displaying graphical text (as in a textual ticker), one for use with Phidget LEDs (as used in the Bus LED Display to be discussed later), and one for use with Ambient Orbs (used in the Motion Monitor and Remote Awareness Displays).

PTK Architecture Summary

The PTK's architecture allows peripheral displays to be designed to display information in different ways based on its importance to the user. In particular, we help designers to manage how a display affects user attention through our support for abstraction, notification levels and transitions.

In addition to managing user attention, we support some standard issues addressed by other similar toolkits such as AROMA [24], the Context Toolkit [4], Real World Interfaces [21], Phidgets [8], and iStuff [1]. In particular, we support access to historical input by components such as abstractors or notification maps, remotely distributed input

Table 1: PTK library components

	Library Component or Widget
Input	• Audio: volume and frequency samples • Camera: image frame gathering; connection support for various cameras • Phidgets: input from Phidget Interface Kits [8]; supports various sensors • Context: hooks to input from the Context Toolkit [3] • Calendar: parses online calendars and sends events regarding schedules • News: parses top online news headlines • Stock: parses online NASDAQ information given company symbols • Weather: parses online weather information (e.g. temperature, wind, *etc*.) • Web page parser: general utility • Serial port communication: general utility
Output	• Ticker Text: displays text that scrolls across a ticker JPanel • Ambient Orb: provides interface for displaying events on an Ambient Orb • Phidget Interface Kit: hooks and functionality for up to six Phidgets [8] • PhidgetLEDOutput: provides hooks and functionality for Phidget LEDs • Phidget Servo: provides hooks and functionality for Phidget Servos
Abstractor	• Voices: takes audio input and determines if voices are present • Telephone: takes audio input and determines if a phone is ringing • Motion: takes camera input and determines how much motion occurred • People counting: takes camera input and counts the people present
Notification Map	• Threshold • Exact match • Contains • Degree of change

and output components across devices, and easy reconfiguration of these components with the help of a discovery system.

LIBRARY COMPONENTS

In addition to our architecture, we developed a set of library components that can be connected to quickly create peripheral displays. Here we present them and describe an example display that utilizes these components. Table 1 shows the PTK library components for input, output, abstraction, and notification levels. Given that peripheral displays can incorporate any of a large variety of components, our library is necessarily limited. Therefore, we have focused on making it very easy to create new library components. A new input, output widget, abstractor, or notification map can each be created by extending a base class and implementing a single method. As mentioned earlier, transition support is architectural, and simply causes `displayEvent()` to be called with different parameters depending on an event's notification level. Thus, there are no transition library elements.

Complex Example Using Library Components

To demonstrate the power of the PTK in enabling easy composition of components and the usefulness of the library, we present a complex example application building upon the Motion Monitor: the Remote Activity Display. The display is designed to run in an office environment to give users a sense of the activity level in a remote location. The user can check activity level from anywhere in the room by glancing at the Ambient Orb™. If the user wants more detailed information, she can bring up her ticker.

An Ambient Orb™ is used not only to display motion, but also to display sound sensed by a microphone. A secondary display, a ticker, runs on the user's desktop and displays additional details. Four abstractors are used. Data from the remote camera is abstracted to measure amount of motion and number of people present. Data from the microphone is abstracted to measure presence of voices or telephone rings.

The Ambient Orb component of the Remote Activity Display differs from the Motion Monitor only in its inclusion of audio as a data source. Like the Motion Monitor, the Remote Activity Display's Orb transitions very slowly between shades of the same color for low amounts of motion, and more quickly for high amounts of motion. The ticker shows any information abstracted from the camera and microphone in textual form. For example, in Figure 4, the ticker is displaying a ring event, by showing the word "ring!" If people are present, the text "<number> people" appears and scrolls across the ticker. When voices are detected, "voices!" appears. When motion is detected, the ticker indicates the amount in text (*i.e.,* "low motion" or "high motion"). For "low motion," it fades in slowly as a change blind transition. For "high motion," it abruptly updates as a make aware transition. If people, motion, and audio events occur simultaneously, the ticker shows all (*e.g.*, "2 people, high motion, ring!") using the highest notification level of the events. When no motion or audio events occur, the ticker displays no message.

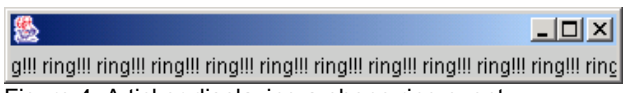

Figure 4: A ticker displaying a phone ring event

This application is entirely composed of PTK library components. The code for creating it is as follows:

```
1  Abstractors[] as = {new TelephoneAbstractor(),
                       new VoiceAbstractor()};
2  Abstractors[] cs = {new MotionAbstractor(),
                      new PeopleCountAbstractor()};
3  InputSource[] ins = {new AudioInput(as),
                        new CameraInput(cs)};
4  Output[]outs= {new ActivityTickerOutput(panel),
                  new ActivityOrbOutput()};
5  String[] audio_ids = { RING, VOICE };
6  a_match = new ExactMatchNotificationMap(
     audio_ids, false, null, true,  null, null);
7  c_match = new ExactMatchNotificationMap(
     AMOUNT_OF_MOTION, MOTION_NONE, MOTION_LO,
     MOTION_HI, null, null);
8  thresh = new ThresholdNotificationSetter(
     NUM, new Range(0,0), new Range(1,999),
     null, null, null);
9  NotificationMap[]map= {a_match,c_match,thresh};
10 makeConnection(ins, outs, map);
```

In lines 1 and 2, telephone, voice, motion, and people counting abstractors are instantiated. In line 3, audio and camera inputs are instantiated. The two audio abstractors are passed into the audio input constructor and the two camera abstractors are passed into the camera input constructor. This causes abstraction to occur on the input side. In line 4, wrappers for the ticker and Ambient Orb output widgets are instantiated. In line 6, an `ExactMatchNotificationMap` is created to set the notification level for all events with ids `RING` and `VOICE`. If an event has that id, and a value of `true`, its notification level is set to *make aware.* Otherwise, its notification level is set to *ignore*. Line 7 is identical to line 3 in the Motion Monitor code, and sets the notification levels for motion events. In line 8 a threshold notification map is created to set the notification level to *make aware* when there are between 1 and 999 people counted, and to *ignore* when there are none. Finally in line 10, all of these components are connected using the `makeConnection` method. This causes events from the two inputs to be passed through the notification maps to the two output widget wrappers.

These 10 lines of code, plus an additional 25 lines to customize the ticker and orb output widgets and their transitions and 7 lines to initialize the GUI frame containing the ticker comprises all the logic required for the Remote Activity Display. Building upon the previously presented Motion Monitor, the PTK enabled us to add considerable functionality with little additional code: two input sources, two output modalities, and specific events based upon features extracted from the audio input. It also allowed us to manage user attention by assigning

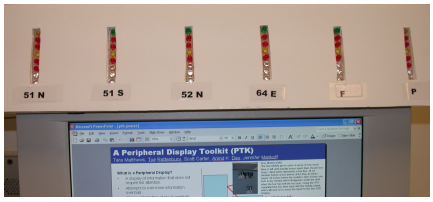

Figure 5: **L:** BusMobile **R**: Bus LED

Figure 6: Social Guitar

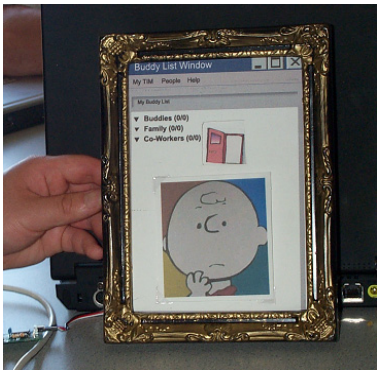

Figure 7: IM Picture Frame

importance (notification levels) to data abstracted from raw video and audio input and to display them appropriately using transitions.

EXAMPLE APPLICATIONS

We have designed and implemented a number of applications using the PTK, in addition to the Motion Monitor and Remote Activity displays. Three additional applications are presented here, all based on displays that had previously been built by researchers at Berkeley without the PTK: a display of bus arrival/departure information demonstrates how our support for managing user attention helped us to improve a display design [18]; a display of ambient audio demonstrates how the separation of concerns in our toolkit allowed us to explore a variety of display designs [10]; and a display of instant messenger status built by a developer not in our group allowed us to get feedback about our toolkit design [3].

Together, these applications demonstrate that the PTK supports separation of concerns and facilitates code re-use, and that it allows applications to be defined in terms of issues relating to human attention. Below we present each peripheral display, followed by a discussion highlighting how they illustrate different aspects of our architecture.

Bus Display

To provide public transit users with an awareness of when popular buses will arrive at their chosen bus stops, we designed two peripheral displays: the Bus Mobile [18], and the Bus LED Display (Figure 5). Both show the same bus schedule input, which indicates the time until bus arrival. However, the Bus LED was designed to improve on the Bus Mobile by addressing some usability problems regarding how it managed user attention.

The Bus Mobile 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. In user studies, we found that this design was problematic because the most noticeable change in the display occurs when the token lowers, 24 minutes before bus arrival. There is no significant change at a time when the user might want to notice the display, shortly before a bus is about to leave.

The Bus LED Display consists of six columns of LED lights controlled by a `PhidgetLEDOutput`, arranged horizontally, labeled below with the corresponding bus number. Each column represents a bus line, and progressively lights up as the bus nears the bus stop. 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. Of course, if the user chooses to look at it, she can determine a bus' location with a glance.

Ambient Audio

Although we may take it for granted, peripheral awareness of ambient sound allows us to manage interruptions and monitor people and devices in both our homes and offices. A peripheral display can provide the same functionality to a person who is deaf, or who wants to monitor a remote location. In the past we have experimented with two different displays that show ambient sounds visually [10]. We used the PTK to explore three additional representations of ambient audio. In addition to the ticker shown in Figure 4, we created an Audio Orb and a Social Guitar (Figure 6). When audio volume levels are low, the Audio Orb pulses green slowly and dimly. As the volume increases, it quickens the pace at which it transitions between them. When the phone rings the Orb pulses red, and when voices are detected it pulses blue. The Social Guitar (Figure 6) provides awareness of audio events occurring in a remote space by plucking a string. A Phidget servomotor in the round 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 the pitch and 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.

Instant Messenger Status

Our final example, like the others, is a re-implementation of existing work. However, in this case, we asked a student not associated with the PTK project to use our toolkit. The IM Picture Frame (Figure 7) is a tangible display of user activity taken from an instant messenger (IM) client [3]. Originally implemented without the PTK, the application

retrieves user status from an IM client, and displays an image at the top of the frame indicating whether the user is present, typing, or idle.

The display was rebuilt with the PTK. When we interviewed our user about his experience using the PTK, he said that it "standardizes the model for creating peripheral displays. I like the PTK model - it made my code cleaner. I think I'll continue to use the PTK to build more applications." His biggest concern was a lack of support for debugging. Currently, the PTK does not have any support for managing or learning about the status and connections between various inputs, outputs, and other components. Despite this, it only took him 4 hours to learn how to use our API and build a display with it.

Discussion

Our applications demonstrate that the PTK facilitates code re-use, supports the design of displays that consider human attention through abstraction, notification levels, and transition, and enables interface experimentation.

The systems we re-created include the BusMobile application and the Ticker and Ambient Orb library output widgets. These systems were originally designed without notification levels and transitions. The PTK helped us to think about the importance of different incoming data, and the best ways to display it appropriately. For example, the Bus LED display improves on the Bus Mobile by ensuring that the most noticeable change occurs when the user would need to leave to catch the bus. In the case of the library elements (the Ticker and Ambient Orb), our added support for transitions and notification levels made their re-use in different applications particularly easy, since they can support default behaviors appropriate to different notification levels.

The ambient audio displays were all new variations on an idea first explored without the PTK [10]. They demonstrate how we can quickly experiment with a variety of designs, all based on the same input, that make use of three different outputs, different notification levels and types of abstraction (volume levels, rings, and voices).

The PTK makes it easier to build displays. A student who re-implemented an application with the PTK was able to quickly learn and use the architecture, and found it helpful in organizing his code. Our own experience was that all of the displays presented here that were created with the help of PTK library elements required between 10 and 40 lines of logic code to construct.

Our examples also demonstrate that the PTK library is complete enough to support the creation of a variety of interesting applications. Library components we used included telephone, voice, and motion abstractors; threshold and exact match notification maps; camera and audio inputs; and Phidget, Ambient Orb, and ticker outputs. Together, we feel that these applications illustrate the flexibility and strength of the characteristics we support in the toolkit.

RELATED WORK

Several toolkits exist that address issues related to the PTK, though none to our knowledge supports abstraction, notification levels, and transitions. AROMA [24] is an architecture that supports building peripheral displays based on abstraction. The Context Toolkit [4] provides an infrastructure for using contextual input in applications and provides some support for abstraction through context interpreters. Neither supports notification levels or transitions. The Notification Platform [11] senses user interruptibility and focus of attention and determines when, on what device, and in what modality, to display notifications. It does not support *how* to display information in an attention-sensitive manner, as the PTK does using abstraction and transitions. Phidgets [8] provide a library of easy to use USB-based hardware input sensors and output widgets and the Real World Interfaces Toolkit [21] provides a library of X10 output widgets, but both these systems leave the job of providing abstraction, notification levels and transitions to the programmer. Finally, both iStuff [1] and the Context Toolkit enable easy reconfiguration of distributed inputs and outputs to support interface experimentation. We build on this past work by providing architectural support for using abstraction, notification levels and transitions to manage user attention, and a library of useful components tailored to peripheral display creation. Additionally, we provide hooks and library components enabling peripheral display developers to leverage inputs and outputs from both Phidgets and the Context Toolkit.

FUTURE WORK

The PTK represents groundwork for future research in peripheral displays. It provides support for designing displays that manage the way they impact user attention. However, its support for managing user attention would be enhanced by knowledge about current user activities and interruptibility. We plan to apply existing algorithms or systems for sensing user interruptibility [6] and other local context [11] to dynamically modify the notification levels assigned to events, and the transitions used to display different events. For example, an audio display like Audio Aura [23] could be quiet when a user is talking with colleagues.

Additionally, although the toolkit has already proven useful in designing a variety of displays, we plan to expand our animation support to include pluggable control over pacing issues [12]. We would also like to address modality specific animation issues such as imperceptibly looping audio sequences.

Future work is also needed to inform peripheral display design. By making it easier to implement displays, the PTK enables design exploration. By providing structure support for displaying information on the periphery of human attention, the PTK helps address aspects of good peripheral display design. However, few lengthy or comparative evaluations of peripheral displays have been conducted from which to guide future designs. As a next step, we plan

to build support for evaluation into the PTK to allow developers to more easily evaluate the displays they build.

CONCLUSION

Peripheral displays are an important class of ubiquitous computing applications. However, to build a display that will be adopted by users, a designer must carefully manage how it impacts user attention. Three particular aspects of displays that affect user attention are abstraction, notification levels and transitions. We developed a toolkit, the PTK, to support peripheral display development, based on these characteristics. In addition to simplifying the construction of peripheral displays by supporting history, component distribution and discovery of available input sources, our toolkit provides explicit, re-usable support for each of the three characteristics we identified.

We have validated this toolkit by building a variety of peripheral displays using the PTK. These displays illustrate the importance of the characteristics highlighted in this paper, and show that by supporting them, the toolkit eases prototyping, enables interface experimentation and supports the design of displays that consider human attention through abstraction, notification levels, and transitions.

REFERENCES

1. Ballagas, R., *et al.* "iStuff: A physical user interface toolkit for ubiquitous computing environments". In *Proc. of CHI '03*, pp. 537-544.
2. Chang, A., *et al*. "Lumitouch: An emotional communication device." In *Extended Abstracts of CHI '01*, pp. 371-2.
3. De Guzman, E. *et al*. "Exploring the design and use of peripheral displays of awareness information." To appear in *Extended Abstracts of CHI '04*.
4. Dey, A.K. *et al.* "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *HCI Journal. 16*, 2-4:97-166. 2001.
5. 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.
6. Fogarty, J. *et al.* "Examining the robustness of sensor-based statistical models of human interruptibility." To appear in *Proc. Of CHI '04*.
7. Gaver, W. T. *et al.* "Effective sounds in complex systems: the ARKola simulation." In *Proc of CHI '91*, pp.85-90.
8. Greenberg, S. and Fitchett, C. "Phidgets: Easy development of physical interfaces through physical widgets." In *Proc. of UIST '01*, pp. 209-218.
9. Heiner, J. M. *et al.* "The Information Percolator: Ambient information display in a decorative object," In *Proc. of UIST '99*, pp. 141-148.
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*, pp. 161-168.
11. Horvitz, E. and Apacible, J. "Learning and reasoning about interruption." In *Proc. of ICMI '03*, pp. 20-27.
12. Hudson, S.E. and Stasko, J.T. "Animation support in a user interface toolkit." In *Proc of UIST '93*, pp. 57-67.
13. Intille, S. S. "Change blind information display for ubiquitous computing environments." In *Proc. of Ubicomp '02*, pp. 91-106.
14. Ishii, H. *et al.* "Pinwheels: visualizing information flow in an architectural space." In *Extended abstracts of CHI '01*, pp. 111-112.
15. Linnett, C. "Perception without attention: redefining preattentive processing." PhD Thesis. UC Berkeley. 1996.
16. Mack, A. "Perceptual organization and attention." *Cognitive Psychology*. 24:475-501. 1992.
17. Mack, A. and Rock, I. *Inattentional Blindness*. MIT Press, Cambridge. 1998.
18. Mankoff, J. *et al.* "Heuristic evaluation of ambient displays." In *Proc. of CHI '03*, pp. 169-176.
19. Matthews, T., *et al.* "A peripheral display toolkit." U.C. Berkeley Tech Report, CSD-03-1258, 2003.
20. McCrickard, D.S., *et al.* "Evaluating animation in the periphery as a mechanism for maintaining awareness." In *Proc. of INTERACT '01*, pp. 148-156.
21. McCrickard, D.S. *et al.* "Supporting the construction of real world interfaces." In *Proc. of Human Centric Computing Languages & Environments '02*, pp.54-56.
22. Miller, T. and Stasko, J. "Artistically conveying information with the InfoCanvas: A highly personalized, elegant awareness display", In *Proc. of AVI '02*.
23. Mynatt, E.D., *et al.* "Designing audio aura." In *Proc. of CHI '98*, pp. 566-573.
24. Pedersen, E. R., and Sokoler, T. "AROMA: Abstract representation of presence supporting mutual awareness." In *Proc. of CHI '97*, pp. 51-58.
25. Plaue, C., *et al.* "Is a picture worth a thousand words? An Evaluation of Information Awareness Displays." To appear in *Proc. Graphics Interface '04*.
26. Posner, M. and Petersen, S. "The attention system of the human brain." *Rev. of Neuroscience*, 13:25-42. '90.
27. Rensink, R. "Change detection." *Annual Review of Psychology*, 53:245-77. 2002.
28. Schmell, R.W. and Umanath, N.S. "An experimental evaluation of the impact of data display format on recall performance," *CACM*, *31*, 5:562-70, 1988.
29. Treisman, A. "Distributed attention." In *Attention: Selection Awareness and Control*, pp. 5-35. 1993.
30. Weiser, M. and Brown, J.S. "Designing calm technology." *PowerGrid Journal, 1*, 1, July 1996.