Oasis: A Framework for Linking Notification Delivery to the Perceptual Structure of Goal-Directed Tasks

15

SHAMSI T. IQBAL
Microsoft Research
and
BRIAN P. BAILEY
Microsoft Research and University of Illinois at Urbana-Champaign

A notification represents the proactive delivery of information to a user and reduces the need to visually scan or repeatedly check an external information source. At the same time, notifications often interrupt user tasks at inopportune moments, decreasing productivity and increasing frustration. Controlled studies have shown that linking notification delivery to the perceptual structure of a user's tasks can reduce these interruption costs. However, in these studies, the scheduling was always performed manually, and it was not clear whether it would be possible for a system to mimic similar techniques. This article contributes the design and implementation of a novel system called Oasis that aligns notification scheduling with the perceptual structure of user tasks. We describe the architecture of the system, how it detects task structure on the fly without explicit knowledge of the task itself, and how it layers flexible notification scheduling policies on top of this detection mechanism. The system also includes an offline tool for creating customized statistical models for detecting task structure. The value of our system is that it intelligently schedules notifications, enabling the reductions in interruption costs shown within prior controlled studies to now be realized by users in everyday desktop computing tasks. It also provides a test bed for experimenting with how notification management policies and other system functionalities can be linked to task structure.

Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*Human Information Processing*; H.5.2 [**Information Interfaces and Presentation:**]: User Interfaces— *Evaluation/methodology, user-centered design*

General Terms: Experimentation, Human Factors, Measurement

Additional Key Words and Phrases: Interruption, attention, notification management systems, breakpoints

This work was supported in part by the National Science Foundation under award no. IIS 05-34462.
Authors' addresses: S. T. Iqbal, One Microsoft Way, Microsoft Corp., Redmond, WA 98052; email: shamsi@microsoft.com; B. P. Bailey, 201 N. Goodwin Ave., Department of Computer Science, University of Illinois, Urbana, IL 61801; email: bpbailey@illinois.edu.

DOI 10.1145/1879831.1879833 http://doi.acm.org/10.1145/1879831.1879833

ACM Reference Format:

Iqbal, S. T. and Bailey, B. P. 2010. Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks. ACM Trans. Comput.-Hum. Interact. 17, 4, Article 15 (December 2010), 28 pages.
DOI = 10.1145/1879831.1879833 http://doi.acm.org/10.1145/1879831.1879833

1. INTRODUCTION

Notifications are being increasingly used as a powerful mechanism for proactively delivering information to users in many multitasking domains [McCrickard et al. 2003]. We use the term *notification* to refer to a visual cue, auditory signal, or haptic alert generated by an application or service that relays information to a user outside her current focus of attention. Notifications can provide useful benefits such as supporting near instant communication [Czerwinski et al. 2000b; Latorella 1999], enabling awareness of peripheral information [Maglio and Campbell 2000], and relaying reminders of upcoming activities [Dey and Abowd 2000]. At the same time, notifications are almost always delivered as soon as the underlying information becomes available without regard for the state of the user's ongoing task. An important challenge is, therefore, to understand how to appropriately balance the timeliness of information delivery with the cost of interrupting user tasks. This article presents the architecture and implementation of a novel system, Oasis, that addresses this challenge by leveraging the perceptual structure of user tasks to mediate notification delivery.

Improved notification management is important for end users, as controlled studies have consistently shown that this type of immediate interruption incurs costs for users in terms of decreased productivity and increased negative affect [Adamczyk and Bailey 2004; Bailey and Konstan 2006; Czerwinski et al. 2000b; Kreifeldt and McCarthy 1981; Zijlstra et al. 1999]. The accumulation of these costs may also be significant, as field studies have found that users receive many notifications during their workday [Iqbal and Horvitz 2007; Jackson et al. 2001]. Recent work has also shown that users prefer to continue the use of notifications to cue them to new information despite the interruption costs rather than to turn the notifications off and have to repeatedly check for new information manually [Iqbal and Horvitz 2010]. The problem of managing interruptions from external sources, through notifications or other mechanisms, is not constrained to the desktop. Similar problems have been identified in other multitasking environments, such as, aviation cockpits [Dismukes et al. 1998; Latorella 1999], in-vehicle systems [Lee et al. 2004] and control rooms [Stanton 1994].

An active thread of research in the domain of interruption management has been to explore techniques for delivering notifications such that the interruption costs are reduced without significantly affecting the timeliness of the notifications [Adamczyk and Bailey 2004; Czerwinski et al. 2000a; Fogarty et al. 2005; Horvitz et al. 1999; Iqbal and Bailey 2005; Iqbal and Bailey 2006]. Prior research has formulated one particular approach where notification delivery is linked to the occurrence of *breakpoints* within a user's tasks. The concept of

a breakpoint originated from research in psychology showing that the human perceptual system segments observed activities into a hierarchical structure of discrete action units [Newtson and Engquist 1976] and that people typically generate similar structures for the same activity [Zacks et al. 2001]. The boundary between two adjacent action units is called a *breakpoint* and breakpoints can be categorized based on the granularity of the units segmented.

For interactive computing tasks, it has been shown that there are at least three granularities of breakpoints—Fine, Medium, and Coarse—that can be reliably detected by users [Iqbal and Bailey 2007]. Empirical studies have shown that deferral of notifications until breakpoints are reached during task execution reduces interruption costs as measured by resumption lag and subjective assessments of frustration [Adamczyk and Bailey 2004; Iqbal and Bailey 2005; Iqbal and Bailey 2006]. One explanation as to why these moments are less disruptive is because they correspond with transient reductions in mental processing effort [Bailey and Iqbal 2008]. Studies have further shown that coarser breakpoints correspond with successively larger reductions in interruption cost [Iqbal and Bailey 2006]. In all of these studies, however, the execution sequences of the experimental tasks were controlled, the locations of the breakpoints within those sequences were identified a priori, and it was the experimenter who manually triggered delivery of the notifications. It was therefore unclear whether a computational system would be able to mimic similar techniques or whether similar reductions in cost could be achieved for free-form tasks, such as, tasks controlled by the goals of the user. Instead of breakpoints, researchers have investigated the use of other aspects of task structure such as the planning, execution, and evaluation stages of a task for notification delivery [Czerwinski et al. 2000a; Monk et al. 2002]. Though the empirical results are of theoretical interest, implementing these approaches would require explicit knowledge of the tasks themselves. A distinct advantage of using breakpoints is that they can be detected without explicit knowledge of the tasks, thereby allowing the technique to be more broadly applied.

Our notification management system, Oasis (Omniscient Automated System for Interruption Scheduling) automates the process of aligning notification scheduling with the occurrence of breakpoints within free-form tasks for the desktop domain. When an application wants to deliver a notification, it sends a request to Oasis specifying the desired deferral policy (e.g., defer to the next Coarse, Medium, or Fine breakpoint). Oasis monitors the user's interaction stream, detects breakpoints on the fly, and signals the application when it identifies a breakpoint satisfying the specified policy, after which the application can render the notification. For example, imagine that a user is cognitively engaged in editing a document and a notification is generated by her email client. Our system allows the notification to be deferred until the user completes the current sentence or paragraph rather than interrupt in the middle of editing. In general, when a user is engaged in a task and a notification is generated, our system allows the user to complete the execution of her current thought (i.e., the operations associated with the currently active "chunk" in memory) before allowing the notification to be delivered. In relation to other prototype systems for notification management, our system is the first to implement a

strategy for scheduling notifications that is grounded in cognitive principles; in this case, principles related to the use of perceptual task structure.

We describe the architecture and implementation of the system, how it detects breakpoints (and implicitly, task structure) without requiring explicit knowledge of the task itself, and how it layers a set of flexible notification scheduling policies on top of this detection mechanism. Oasis is designed to identify breakpoints within goal-directed interactive tasks such as document editing, image manipulation, and programming tasks, and provides access to three levels of breakpoints: Coarse, Medium and Fine. By detecting three breakpoint granularities, Oasis offers increased flexibility in balancing interruption cost with the timeliness of notification delivery. Oasis detects Coarse breakpoints independent of any specific application by tapping into the already available system-level event stream. Detection of Medium and Fine breakpoints requires access to an application's event stream, which can typically be exposed using lightweight plug-ins. For demonstration purposes, the current implementation of Oasis provides plug-ins for two commonly used applications, Microsoft Visio (diagram editing) and Visual Studio (programming). Plug-ins for additional applications can be developed independently and interfaced with Oasis. To support these applications, situations where improved breakpoint detection is needed, or for other research purposes, the system includes a data collection module, data annotation tool, and learning component that allows statistical models for breakpoint detection to be created or refined.

The contribution of this article is the description of the architecture, implementation, and usage scenarios of a functional notification management system. The system demonstrates the feasibility of automating the notification scheduling techniques that were manually applied in our prior empirical studies for controlled tasks, thereby enabling similar results to now be realized for desktop computing tasks. The system also provides a test bed for exploring how different notification management policies and other system functionalities (e.g., automated structuring of command histories) can be linked to perceptual task structure. Although we have previously reported empirical results for how the use of our system impacts users and their tasks [Iqbal and Bailey 2008], this article focuses on a discussion of the architecture and implementation details of our system and the rationale for its design.

2. RELATED WORK

In this section, we discuss the benefits and costs of notification, the theory of perceptual task structure and its efficacy for notification scheduling, and other systems and strategies for managing notifications.

2.1 Benefits and Costs of Notifications

An application uses notifications to proactively deliver information to the user, rendered through one of several sensory modalities—visual, auditory or haptic [Lee et al. 2004; McCrickard et al. 2003]. Perhaps the most salient benefit of using notifications is that they reduce the user's need for repeatedly checking an external information source, a behavior which is also detrimental to the user's

ongoing task [Maglio and Campbell 2000]. The use of notifications can be beneficial in many situations, such as, to support peer communication [Czerwinski and Horvitz 2002], provide awareness of collaborator activities [Dabbish and Kraut 2004], and relay application assistance at appropriate moments [Maes 1994].

At the same time, however, current applications almost always deliver notifications as soon as the underlying information becomes available. This often results in the user being interrupted in the midst of an ongoing action. Researchers have studied this phenomenon extensively and found these types of interruptions result in reduced task productivity [Czerwinski et al. 2000a; Kreifeldt and McCarthy 1981; Latorella 1998; McFarlane 1999; Monk et al. 2002], impaired decision making [Speier et al. 1999], and increased negative affect [Adamczyk and Bailey 2004; Zijlstra et al. 1999]. For example, Bailey and Konstan [2006] have shown that notifications interrupting the current task cause users to take up to 27% more time to complete the task, commit up to twice the errors, and experience up to twice the anxiety. Gillie and Broadbent [1989] showed that these types of disruptive effects can be caused by the complexity of the interrupting task and its similarity to the main task. Recovery and resumption of interrupted activities also take substantial time, between 10–25 minutes, and this has been corroborated by several studies [DeMarco and Lister 1999; Gonzalez and Mark 2004; Iqbal and Horvitz 2007]. Interruption costs are often attributed to limitations in human information processing capabilities, where the additional demands on cognitive resources due to the interrupting task interfere with and reduce performance on the primary task [Navon and Gopher 1979; Wickens 2002].

Our research aims to achieve a more flexible and effective balance between the costs and benefits of notifications for end users. In our approach, notifications are deferred for a short time in exchange for a meaningful reduction in the ensuing interruption cost. This is achieved by linking notification delivery to the perceptual structure of user tasks.

2.2 Perceptual Task Structure and Its Use for Notification Management

Perceptual structure refers to how the human perceptual system segments an observed, goal-directed activity into discrete units [Newtson 1973]. The boundary between two adjacent units is called a *breakpoint*, which can be thought of as a moment of transition in perception or action [Zacks and Tversky 2001]. Controlled experiments have shown that human observers generally agree on the location of the breakpoints within a given goal-directed activity, indicating there is a shared cognitive schema that drives the perceptual system [Zacks et al. 2001]. Observers have reported using cues such as the completion of an action, change in pace of the action, and change in the object of focus as salient indicators of a breakpoint [Zacks and Tversky 2001]. Zacks et al. [2001] offered evidence showing that the perceptual system organizes observed activity into at least a two-level hierarchical structure. This was achieved by asking independent observers to watch video recordings of other people performing physical tasks, such as, folding clothes, washing dishes, making beds etc., and

Table I. Definitions and Actions Corresponding to the Different Types of Breakpoints

Breakpoint	Definition	Example of corresponding action
Coarse	transition from the largest meaningful and natural unit of action to the next	Switch from an ongoing programming task to interacting with a media application
Fine	transition between the smallest meaningful and natural unit of execution to the next	Switch from actively editing code to compiling and debugging
Medium	transition between natural and meaningful units which are smaller than Coarse but larger than Fine	Switch from one source code file to another within the same project

mark the locations of breakpoints that separated the largest meaningful units of action (defined as *coarse* breakpoints) and those that separated the smallest units (defined as *fine* breakpoints); and then analyzing their temporal alignment. Results showed that the observers generally agreed on the locations of coarse and fine breakpoints (within 1sec windows) and that the coarse unit boundaries aligned with the fine unit boundaries more often than would be predicted by chance. These results provide evidence of the existence of hierarchical structures in physical tasks and that these structures can be reliably detected by observers who did not have the experience of performing the task themselves.

Building on this corpus of theoretical work in the domain of interactive computing tasks, Iqbal and Bailey [2007] showed that observers can reliably identify three granularities of breakpoints within task execution sequences. In addition to the coarse and fine granularities identified by Zacks et al. [2001], an additional granularity, *medium*, was also identified, indicating transitions between units smaller than those surrounding coarse breakpoints but larger than those surrounding fine. For example, for a user engaged in programming, switching from the programming activity to interacting with a media player (or other unrelated task) can indicate a Coarse breakpoint. A Fine breakpoint can occur by switching from editing code to compiling and debugging the code while a medium breakpoint can occur when switching between two source files to edit independent sections of code. Table I defines the three granularities of breakpoints for interactive tasks and summarizes these examples. Examples of breakpoints for other task domains are analogous. As with Zacks et al.'s [2001] work, Iqbal and Bailey [2007] also found that the locations of breakpoints identified by observers were mostly consistent with the locations identified by the users who performed the tasks. This agreement may be explained because individuals are also observers of their own actions.

Inspired by Miyata and Norman's [1986] conjecture that the transitions between different tasks and phases of a task would represent less disruptive moments for interruption, our prior work has tested the efficacy of using breakpoints for notification scheduling. Through several controlled studies, our results have shown that delivering notifications at breakpoints results in lower interruption costs than when delivered immediately [Iqbal and Bailey 2005; Iqbal and Bailey 2006]. Results further showed that coarser granularities of breakpoints correspond with successively lower interruption costs [Iqbal and

Bailey 2006]. Similar results have been shown to hold for different task domains, including document editing, programming, diagram editing, and image manipulation, which argues that the technique is reasonably general. In these experiments, however, the execution sequences of the experimental tasks were controlled, the locations of the breakpoints were manually defined a priori, and it was the experimenter who triggered the delivery of the notifications using a Wizard of Oz paradigm.

Other researchers have also leveraged task structure to investigate more effective strategies for scheduling notifications. For example, Czerwinski et al. [2000a] showed that interruptions during the execution phase of a task compared to the planning or evaluation phase cause users to take more time to switch to the interrupting task. Monk et al. [2002] demonstrated that interrupting before the beginning of a subtask causes less resumption lag than interrupting during a subtask. This corpus of empirical studies has typically leveraged different phases of a task whereas our work emphasizes the use of specific moments within an execution sequence such as breakpoints. The advantage of using breakpoints is that they exist in nearly all goal-directed tasks and can be detected without explicit knowledge of the tasks themselves.

This article contributes the design of a system that automates the process of linking notification scheduling to the occurrence of breakpoints within real (uncontrolled) user tasks. Our system leverages the use of statistical models for detecting the three granularities of breakpoints defined in Table I [Iqbal and Bailey 2007], but significantly extends that prior work by layering a set of flexible deferral policies on top of the models and by implementing a complete end-to-end process for scheduling notifications. The system also provides a separate tool set for building customized models for detecting breakpoints within user activities.

2.3 Strategies and Systems for Managing Notifications

Notification management generally requires coordination between the interrupter and interruptee to effectively balance the cost and benefit of notifications. Existing systems can therefore be categorized as supporting at least one of three strategies; aiding the interrupter, aiding the interruptee, or supporting the coordination between them. The behavior of Oasis falls primarily into the third category. However, it could benefit from having access to the information that other systems provide to aid the interrupter, and its output could be utilized by other systems or rendering techniques meant to aid the interruptee.

For aiding the interrupter, systems such as Lilsys [Begole et al. 2004], MyVine [Fogarty et al. 2004], and ConNexus [Tang et al. 2001] provide visual cues of the interuptee's availability to be used by the interrupter. Examples of such cues include office presence, social engagement, and desktop activity. The interrupter can assess these cues to determine an appropriate moment for initiating communication. In relation to our system framework, similar cues could also be accessed by applications to determine an appropriate time window (see Sections 3.1 and 4.2) that would accompany a specific notification request.

To aid the interruptee, one class of solutions aggregates and renders the notification content within a peripheral display. For example, in the Scope system, communication events, calendar appointments, and system alerts are summarized in a persistent radial display [Van Dantzich et al. 2002]. A drawback is that the interruptee must decide when and how often to check the display for updated information. Another class of solutions renders the notification in a manner that reveals important attributes of the underlying notification content. For example, Gluck et al. [2007] showed that linking the content's utility to the salience of the notification display resulted in better performance and user satisfaction during interactive tasks. In contrast, our system schedules notifications at less disruptive moments during a user's ongoing task. However, because our system only signals the requesting application as to when such a moment occurs, the application can use any modality (e.g., visual, auditory or haptic), level of saliency, and display system for rendering the notification.

Facilitating the coordination between the interrupter and interruptee is the strategy that most closely reflects the one realized within the Oasis system and is representative of what McFarlane calls a *mediated* approach [McFarlane and Latorella 2002]. Systems in this category have typically employed decision theoretic approaches for deciding when to interrupt [Horvitz et al. 1999]. The systems typically use as input some combination of desktop activity, calendar events, and visual and acoustical information from the surrounding task environment. For example, in the Priorities system [Horvitz et al. 1999], Lookout [Horvitz 1999], and the Notification Platform [Horvitz et al. 2003], Bayesian networks were used for inferring a user's focus of attention. This information was used to deliver notifications when the computed utility was high, that is, when the benefits of delivering the information outweigh the cost of interruption. A descendant of the Notification Platform, Bestcom, considered the user's social and task context, current and future availability, and communication preferences to predict the best device and modality for interpersonal communication [Horvitz et al. 2002]. In context of desktop computing tasks, [Fogarty et al. 2005] demonstrated that application events could be leveraged to reasonably predict whether a user is interruptible or not, though they did not build a system to demonstrate the effectiveness of their approach.

This class of systems is potentially powerful, but the decisions made by these systems for when to interrupt have not yet been empirically shown to correspond with reduced interruption costs. In addition, the rationales motivating these approaches have not had firm grounding in cognitive principles, which might make it difficult to interpret or explain the empirical outcomes. Relative to these systems, our system ties notification delivery to the perceptual structure of a user's task and leverages interaction events to automatically detect this structure without specification of the task itself. This technique is grounded in cognitive principles and has been shown to yield favorable results in several empirical studies (see Section 2.2).

Finally, at least one prior system, PETDL, has shared the goal of linking notification delivery to the structure of end-user tasks [Bailey et al. 2006]. That system provided a language for specifying interactive tasks and a separate component for monitoring a user's execution through those tasks. Within a task

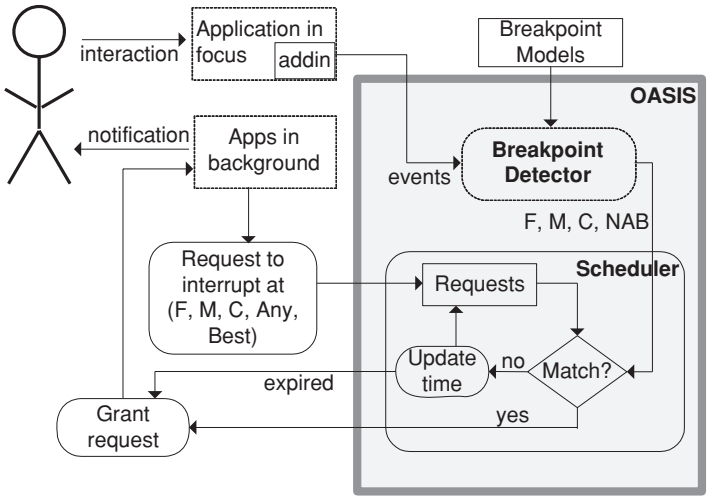

Fig. 1. Schematic of Oasis and its operation (highlighted). This article focuses on describing the details of the design and implementation of the system.

description, costs of interruption could be assigned for each step of a specified task, which could be used for reasoning about when to interrupt. However, manually authoring these types of specifications for free-form tasks, such as those commonly performed in the desktop domain, is complex and requires substantial effort. This would be a major barrier for large-scale deployment of such an approach. In contrast, OASIS automatically detects interruptible moments (i.e., breakpoints) without requiring any explicit specification of the tasks, thereby removing this barrier.

3. OASIS

Oasis is a system service that schedules notifications to occur at perceptually meaningful breakpoints during user tasks. A schematic of the system is shown in Figure 1. Oasis acts as an intermediary between the user and applications that want to deliver notifications. The following scenario illustrates how Oasis functions to better manage notifications.

Imagine a user cognitively engaged in designing a complex diagram using Microsoft Visio or other appropriate tool. As the user is interacting with the diagram, a collaborator sends a mail message detailing several additional features that need to be included in the diagram. Rather than notify the user immediately and risk disrupting the user's ongoing actions, the mail client sends a notification request to Oasis. The request specifies that the notification should be deferred until the next fine breakpoint, but should not be delayed more than five minutes. Meanwhile, the user continues to add and manipulate elements in the diagram and Oasis continues to analyze this stream of actions. When the user completes a particular chunk of actions (e.g., inserts and then sets the properties on a new shape), Oasis identifies this moment as a fine breakpoint and compares it against the policy specified for the notification.

Since it matches, Oasis signals the mail client and the client can then render the notification using its preferred technique.

In this scenario, because the user is able to complete her immediate thought (actions) before being interrupted, she can switch her attention to the notification and resume the interrupted task with more efficiency and less frustration than if it had been delivered immediately. In this latter situation, the notification would have likely interrupted an ongoing action. Had the user turned off notifications, she may have missed information relevant to the task. Finally, if a matching breakpoint did not occur within the specified time window, Oasis would have granted the request at the specified time of expiration.

3.1 Policies for Scheduling Notifications

The scheduling behavior of Oasis reflects a mediated interruption management approach where the system acts as an intermediary between the user and the interrupting application and determines appropriate moments for delivering notifications [McFarlane and Latorella 2002]. This approach is implemented within Oasis through a set of scheduling policies. To send a notification to the user via Oasis, an application prepares a notification request consisting of the desired scheduling policy and maximum time it can wait and sends this request to Oasis. The policy specifies the type of breakpoint at which the notification is to be delivered. As a starting point, Oasis offers five policies giving different balances between notification timeliness and interruption cost that applications can specify in their request:

Next Coarse. This policy specifies that the notification is to be delivered when the user reaches the next Coarse breakpoint in their interaction sequence. A Coarse breakpoint is the moment between two units of action perceived to be the largest, meaningful units in the given context [Iqbal and Bailey 2007]. For example, in the desktop domain, a Coarse breakpoint could be the transition from a programming activity to interacting with a digital media player. As Coarse breakpoints typically occur less frequently but correspond with lower interruption costs, this policy can be utilized for notifications of general interest to the user but that are not urgent or relevant to the ongoing task. For example, a reminder about an upcoming talk might be best scheduled using this policy.

Next Fine. This policy specifies that the notification is to be scheduled at the next Fine breakpoint. A Fine breakpoint is defined as the moment between two units of action considered to be the smallest meaningful units in the given context. An example of a Fine breakpoint is when the user switches from editing source code to starting a compilation. This policy is useful for notifications relevant to the current task or otherwise cannot be deferred for too long. For example, a notification for an instant message describing an error in the source code the user is working on might choose to use this policy. This ensures that the user receives the information in the context of the ongoing task but with less disruption compared to delivering it immediately.

Next Medium. This policy schedules the notification at the next Medium breakpoint. A Medium breakpoint is the moment between two units of action where those units are smaller than Coarse but larger than Fine. An example

Table II.
Summary of the deferral policies along with heuristics for selecting each policy based on the notification content and desired outcome for the user.

	Notification content is ...		Desired outcome	
Policy	Relevant	Urgent	Less disruption	Faster delivery
Next Coarse			√	
Next Medium	√		√	
Next Fine	√	√		√
Next breakpoint of any type		√		√
Coarsest breakpoint in a given timeframe			√	

is when the user has just completed browsing an online API reference and is resuming the editing of source code during a programming task. This policy is useful for notifications that are relevant to the ongoing task but not so urgent that they need to be delivered at a Fine breakpoint. For example, a mail notification from a collaborator about an issue with a related programming project could be deferred until a Medium breakpoint.

Next breakpoint of any type. This policy specifies that the notification is to be scheduled at the next breakpoint of any type: Coarse, Medium or Fine. This policy ensures that a notification is delivered at the next perceptual break, regardless of its granularity. This policy would be useful for notifications that have increased urgency for the user. For example, a notification about a required meeting that is starting in a few minutes might be best scheduled using this policy.

Coarsest breakpoint in a given timeframe. This policy specifies the notification is to be scheduled at the breakpoint that has the lowest interruption cost (i.e., coarsest breakpoint) in a given timeframe. On detection of a breakpoint, the system uses the frequency of past breakpoints and their type to predict the likelihood of receiving a coarser breakpoint within the remaining time. If the likelihood is high, then the system waits. Otherwise the request is granted at the current breakpoint. Note that the previous policy emphasizes determining the *first* interruptible moment, whereas this policy emphasizes identifying the moment with the *least* disruption.

The first three policies were derived directly from prior empirical work investigating effects of delivering notifications at different granularities of breakpoints [Adamczyk and Bailey 2004; Iqbal and Bailey 2005; Iqbal and Bailey 2006]. The latter two policies were added to offer more flexible system behavior and were derived from envisioning various application scenarios as well as our experience gained from prior work. Table II summarizes the deferral policies currently offered in our system along with heuristics for selecting each policy based on the notification content and desired outcome for the user.

This set of policies provides only a starting point and should not be considered exhaustive or, in the case of the latter two, mutually exclusive in terms of the breakpoints selected. The policies may be refined and new policies may be added as experience is gained with the system. This experience would also be useful for more carefully recommending when each policy is best applied. We

assume that applications will choose appropriate policies guided by the criteria in Table II or other relevant information. In Section 4.3, we discuss how existing systems can be used or Oasis might be extended to provide additional information useful for selecting an appropriate policy.

An application can specify a maximum time to wait as part of its notification request in order to prevent overly long deferrals, similar to the concept of bounded deferral discussed in Horvitz et al. [2005]. The timeframe provides an upper bound on the time within which the notification must be delivered so that it does not become irrelevant for the user. If no breakpoint occurs by the end of the given timeframe, the notification request is granted immediately. This situation should be rare but, if it does occur, the resulting system behavior (i.e., interrupt now) should be no worse than today's interface. We assume that a notification delivered at any moment within the given timeframe has equal utility for the user and applications can determine what an appropriate timeframe should be. Heuristics for selecting a timeframe are discussed in Section 4.2.

3.2 Scheduling Notifications

As shown in Figure 1, two runtime components within Oasis, the Scheduler and the Breakpoint Detector, manage notification requests. These components coordinate to (i) identify breakpoints by evaluating user actions against statistical models of breakpoints; (ii) manage notification requests from end-user applications; and (iii) match scheduled requests to the appropriate breakpoints and manage the specified timeframes.

3.2.1 *Detecting Breakpoints.* Detecting breakpoints is necessary for scheduling notifications within Oasis and this process is handled by the Breakpoint Detector: (see Figure 2). The Breakpoint Detector buckets global system and application events, evaluates each bucket of events using a set of statistical models, and outputs a stream of breakpoint and not-a-breakpoint (NAB) decisions. A default set of statistical models is provided with Oasis, but the system allows new models to be created.

Two types of events are received by the Breakpoint Detector: global system events, including mouse, keyboard, and top-level window events, and application events, such as the event generated when text is added to a shape on the canvas in MS Visio. Global events are captured using the Win32 API and the necessary instrumentation is provided by Oasis. However, applications must be instrumented to send their events to our system. For purposes of demonstration, we developed plug-ins for two commonly used applications, Microsoft Visual Studio (source code editing tool) and Microsoft Visio (diagram editing tool) that expose their event stream to our system. Similar plug-ins can be built for other applications.

The default models used within OASIS were learned from the data and process reported in our prior work [Iqbal and Bailey 2007; Iqbal and Bailey 2008]. These models were created by collecting authentic task execution data in the domains of programming and diagram editing, having observers annotate the perceived breakpoints, and learning the mappings from the surrounding

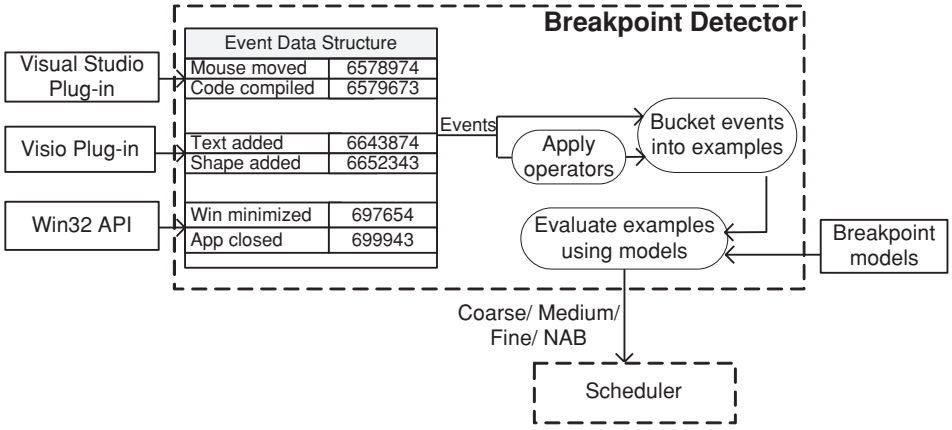

Fig. 2. Schematic of the Breakpoint Detector. Visual Studio and Visio are shown as example applications. Events are bucketed into examples and these are then evaluated using the global model for Coarse breakpoints and the relevant application specific models for Medium and Fine. The decision output (Coarse/Medium/Fine/NAB) is passed to the Scheduler.

interaction data to the identified breakpoints. Further details of how the models were built and their performance are discussed in Section 4.1.

Oasis uses one model for detecting Coarse breakpoints. This model accepts as input the global system events captured by our system and is therefore independent of any specific application. There is an additional model for detecting Medium and Fine breakpoints within each of our demonstration applications. The use of default models eliminates the need for end users to engage in a lengthy training process prior to first use of the system. However, if needed, the models could be trained on a per user basis to improve their accuracy using our offline tool set (see Section 3.3).

When an observed event occurs, it is sent to the breakpoint detector. The breakpoint detector maintains a 30-second history of events and evaluates these events every few seconds using the models. These models map the events to each type of breakpoint and NABs. If a breakpoint is detected, the Scheduler is notified of the occurrence and type of the breakpoint; otherwise, the current moment is reported as a NAB.

3.2.2 *Managing Notification Requests.* When an application wants to render a notification to the user, it first sends a request to the Scheduler via a network connection (see Figure 1). As described in Section 3.1, a request consists of a scheduling policy and a time window. For example, the request (300, next-medium) indicates a defer-to-next-medium policy and a time window of 300s. The Scheduler queues the request, along with the handle to the application, and then waits either for an appropriate breakpoint to occur or for the time window to expire. Once either occurs, the Scheduler signals the application that its request has been granted. The application can then render the notification using the modality and saliency of its choice. Figure 3 gives pseudocode outlining this process.

```
AddInterruptionRequest (Int policy, Time maxWait, Handle connectionhandle)
{
        newRequest.policy ← policy                                  /*create Request,
        newRequest.maxTime← currentTime + maxWait                   include the policy,
        newRequest.AppHandle ← connectionHandle                     max wait time, and
                                                                    application handle
                                                                    */

        requestQueue.add(newRequest)                                /* add request to
                                                                    the queue, sorted
        If requestQueue.First equals newRequest {                   by wait time. If the
           timer.Stop();                                            request is first in
           timer.Interval = maxWait;                                the queue, set the
           timer.Elapsed =  new ElapsedEventHandler(ChkExpired);    timer to its wait
           timer.Start();                                           time */
        }
}

ChkExpired ()
{
        for each i in requestQueue {                                /* Grant expired
          if i.maxTime <= currentTime                               requests and
            lock {                                                  remove from queue
                  send render signal through i.AppHandle            */
                  remove i from requestQueue
             }
        }
        if requestQueue is not empty {
           timer.Stop();                                            /* reset timer to
           timer.Interval =                                         the wait time of
                 notificationQueue.First.maxTime – currentTime;     the first request in
           timer.Elapsed =  new ElapsedEventHandler(ChkExpired);    the queue */
           timer.Start();
        }
}

BreakpointDetectedCallback(Breakpoint breakpoint)                   /* check if
{                                                                   breakpoint satisfies
        for all i in requestQueue                                   any existing
           if breakpoint satisfies i.policy                         requests. If so,
           lock {                                                   grant the requests
                  send render signal i.AppHandle                    and remove from
                  remove i from requestQueue                        queue
           }                                                        */
}
```

Fig. 3. Algorithm (in pseudocode based on C#) for managing notification requests.

When notified of a breakpoint occurrence, the Scheduler checks whether the breakpoint matches any of the policies specified in the requests in its queue. If there is a match, a signal is sent to the corresponding application, which in turn, can render its notification for the user. Matching the policies for defer-until-next-coarse, -medium, -fine or –any-type is straightforward. The most complex policy is defer-until-the-coarsest breakpoint within a given time frame, which we elaborate here.

For this policy, if a Coarse breakpoint is detected, the request is immediately granted since no coarser breakpoint is possible. If the breakpoint is Medium or Fine, the scheduler forecasts the probability of the user reaching a coarser breakpoint within the remaining timeframe. The scheduler calculates this probability as follows:

$$P(\text{coarser breakpoint occurring within X sec} \mid \text{current breakpoint type}) = \#(\text{current to coarser pairs with distance} <= X)/\#(\text{all current to coarser pairs}).$$

For example, suppose the current breakpoint type is Medium and there are 50 seconds left within the timeframe specified for the corresponding request. The system traverses the distribution history of breakpoints to count the number of Medium-to-next-Coarse pairs with a temporal distance less than 50s. This number is then divided by the total number of Medium-to-next-Coarse pairs in the history. If the resulting probability is less than a specified threshold then the system grants the request. Otherwise, it waits for the next breakpoint and the process repeats, until either a coarser breakpoint is not likely or the timeframe given with the request expires. The probability threshold can be specified in a system configuration file.

A default distribution history is included with Oasis. The history contains the temporal distances between each pair of breakpoint types (e.g., Coarse-to-next-Coarse, Coarse-to-next-Medium, etc.). The history is stored as an XML file and is loaded at system startup. The default history was created from the data used to train the default statistical models for detecting breakpoints (see Section 3.2.1). However, if more accurate predictions are needed, our system could be extended to track and use a running history of breakpoints detected for the specific user.

3.3 Tool Set for Developing Breakpoint Models

As described in the previous section, the Breakpoint Detector relies on the use of statistical models for detecting breakpoints. Oasis maintains a set of default models and we expect that the use of default models will be able to suffice for most end users and work contexts. However, for situations where improved performance is desired or for other research purposes, customized models can be created using a model development component within Oasis. The model building process involves recording user interaction data, annotating the breakpoints within the recorded data, and learning the mappings from the interaction data to each type of breakpoint. Oasis provides a set of tools supporting this process which we elaborate in this section.

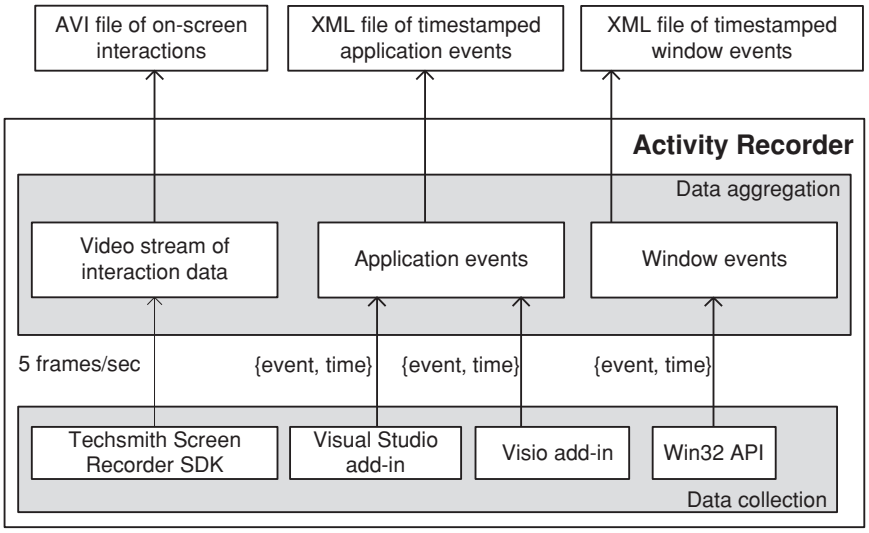

Fig. 4. Schematic of the Activity Recorder service. The data is collected through system APIs and application add-ins, is stored as AVI and XML files, and these files are later accessed by the Breakpoint Annotator.

Model development in Oasis is similar to the process described with the Subtle toolkit [Fogarty and Hudson 2007]. However, Subtle relies on the method of experience sampling for labeling the training data. This method prompts the user to enter the labels at random moments or at moments that can be specified a priori. Because learning models of breakpoints requires the labels to be placed at precise moments within the task execution data and these moments cannot be known a priori, it would be difficult to directly apply this toolkit for learning the models. Our system is therefore different in that it provides a set of tools for capturing and retrospective labeling of the interaction data.

3.3.1 *Recording Interaction Data.* The Activity Recorder is a service used to record interaction data and its architecture is outlined in Figure 4. It collects three categories of data: a video of onscreen interaction, application events, and global system events. The videos are used for retrospective labeling of the breakpoints. The application and global system events are collected and used in raw form as well as to generate higher-level features that may be predictive of the breakpoints. The recorder has a user interface for starting, pausing, and stopping the data collection.

3.3.2 *Labeling Breakpoints.* Once the activity data has been collected, the next step is to annotate the locations of the breakpoints. The Breakpoint Annotator is an interactive tool enabling this functionality, shown in Figure 5. The tool allows a user to view a video of on-screen interaction and annotate where they feel the breakpoints occur. When the user wants to enter a breakpoint, she opens the breakpoint dialog by selecting the appropriate button from the playback control (bottom left of Figure 5). This opens a nonmodal dialogue (bottom

Fig. 5. The Breakpoint Annotator is an interactive tool for marking the locations of breakpoints and identifying their type, entering rationale for the breakpoints, and saving the annotations to a structured file. It also provides a visual summary of the ongoing annotations (right side).

middle of Figure 5) where the user selects the type of the breakpoint and enters the rationale for why she believes this is a breakpoint, if instructed. From within the dialog, the user can scrub the video in different increments to locate the precise time point of the breakpoint. The breakpoint type, its rationale, and the video time point are saved and appear in the summary panel (right of Figure 5). Once complete, the breakpoint data can be saved as a zip archive containing the interaction video, the breakpoint annotation records, and the distribution history of the breakpoints. The latter is used for the prediction algorithm described in Section 3.2.3.

3.3.3 *Learning the Models.* To learn the breakpoint models, the system first creates a set of training examples by associating the global and application event data with the labeled breakpoints. Note that this is a fully automated process and does not require any input from the user. To create training

examples, the system creates a thirty second history of the interaction data preceding each labeled breakpoint. Features are derived from the interaction data and corresponding training examples are generated. Some example features include switching to a mail client (global), finish building project (programming) and finish adding a shape to the current diagram (diagram editing). To distinguish non-breakpoint moments from breakpoints, additional training examples are generated from a random sample of moments that were not identified as breakpoints.

Using the training examples, the learning module filters features that are most predictive and contribute to improved accuracy. A reduced number of features is useful for improving the efficiency of the overall system. The training examples and predictive features are then passed through a set of statistical models based on different learning algorithms including Decision trees, Bayesian nets, Multilayer Perceptrons, and IB1. The models with the highest accuracy are selected and stored in the appropriate location for use by the Oasis system. Further details of this learning process can be found in Iqbal [2008].

3.4 Implementation and Interfacing with Applications

Oasis is fully functional and was developed in Visual C# using the.NET Framework. It consists of about 14,000 lines of code. The runtime component leverages two classes of plug-ins for accessing the user interaction data; a system plug-in and application plug-ins. The system plug-in leverages the Win32 API for capturing events related to the mouse, keyboard, and top-level window manipulations. This plug-in is included as part of Oasis.

Application plug-ins provide access to application-level events and our system relies on others (e.g., end users, developers, or researchers) to create these plug-ins. However, to provide proof of concept, we developed plug-ins for two commonly used applications, Microsoft Visio and Microsoft Visual Studio. The plug-in for Visio was developed using the Microsoft® Visual Studio® Tools for Microsoft Office. This technology allows "add-ins" to be created for any application in the Microsoft Office suite. The plug-in captures events related to manipulating shapes (creating, sizing, deleting, setting properties, etc.), the canvas view position and scale, and manipulating diagrams (creating, naming, saving, closing, etc.). About 500 events are captured with this plug-in. For Visual Studio, the plug-in was developed by extending the IDE. The plug-in reports events related to the editing of the code (ended line of code, method, or class), selection of menu and toolbar commands, and navigation (opening, switching, and closing source files and scrolling the current source file). About 370 events are captured using this plugin. Because application scripting technologies have become widely available, similar plug-ins can be created for a wide variety of end user applications in the desktop domain.

For model development, videos of onscreen interaction are captured using the Techsmith Screen Recorder SDK. The event data is collected from the application and system plug-ins and this data is saved as XML files. The event data and screen interaction data contain time stamps from the system clock allowing for precise synchronization. This is needed when aligning the annotated

breakpoints to the interaction data during the model building process. The breakpoint models are learned using the Weka toolkit [Witten and Frank 2005].

The current implementation of Oasis provides an interface for applications to connect to the Scheduler through an IP Version 4 stream socket using the TCP/IP protocol. For an application executing on the same machine as Oasis, the connection is established using the loopback address (127.0.0.1), but this can be replaced by a different IP address if the application is executing on a different machine. Applications that want to send notification requests set up connections with the Scheduler a priori. The connection is attempted using a nonblocking system call (send) to a predetermined system port. This prevents the application from blocking indefinitely on the request and allows the application to take appropriate action if the connection fails. If successful, the application uses the connection for sending all notification requests to Oasis. Any application capable of sending and receiving messages through IPV4 sockets can interface with Oasis.

4. DISCUSSION

In this section, we review results from a user evaluation of our system and then discuss issues related to its implementation and the generalizability of the underlying technique.

4.1 Evaluation of System Performance and User Impact

We conducted a user study to test the effectiveness of using default statistical models for detecting breakpoints for novel interaction data (i.e., data that was not part of the original training set) and to evaluate how scheduling notifications with our system impacts users and their tasks. Because details of the study have been reported elsewhere [Iqbal and Bailey 2008], we only summarize the methodology and our most significant findings.

The first phase of the study evaluated how well the system could identify and differentiate breakpoints for novel interaction data using the default statistical models. The study was conducted in the task domains of programming and diagram editing. To build the default models, we recruited six users (three per domain) who performed their own tasks in the respective domain. The users had complete control over the task and how it would be performed. Our Activity Recorder (see Section 3.3) was installed on the users' machine and, while the desired task was being performed, collected about 1.5 hrs of interaction data. Examples of the tasks performed included developing a Web-based application in ASP.net and developing information architecture diagrams for a Web site design. Twelve observers viewed the interaction videos and identified the location of breakpoints and their type using the Breakpoint Annotator. The breakpoint data was fed into our model building component which generated the default models. A 10-fold cross-validation showed reasonable performance; recall was 71% for Coarse, 84% for Medium, and 96% for Fine for the programming tasks and 56% for Medium and 58% for Fine for the diagram editing tasks.

To evaluate the models for novel interaction data we recruited six additional users (again, three per domain) who installed and ran our system while they

performed their own tasks in the respective domain. Our system was using the default models and was configured to record the detection of breakpoints. The users later reviewed their interaction videos and retrospectively identified the locations of their breakpoints. The locations of these user-identified breakpoints were compared to the locations of the system-identified breakpoints. Results showed that users and the system agreed on the location of 43% (programming) and 40% (diagram editing) of the breakpoints. However, the default models particularly struggled to differentiate the type of the breakpoints, with recall values ranging between 2 and 42%. These results were lower than expected and highlight the challenge of creating a robust set of default models for breakpoint detection.

We see several directions forward for improving model performance. One is to train the default models using a much larger training set, which might be possible by collecting labeled data from many users or observers. Another direction would be to have users train personalized models, which has been shown to increase model performance [Fogarty and Hudson 2007]. A third direction would be to begin with the default models but train them further with additional data collected from the user.

Given the active research on modeling user activities [Fogarty and Hudson 2007; Horvitz et al. 2003; Horvitz et al. 2002], we believe performance for default and/or personalized models will reach acceptable thresholds for breakpoint detection. We therefore wanted to test the effects of scheduling notifications at breakpoints for free-form user activities. To compensate for the model accuracies, we combined the breakpoint data and learned new models that would detect the occurrence of a breakpoint without differentiating its type. This yielded recall accuracies of 59% and 52% for programming and diagram editing, which was sufficient for the next phase of our study. To identify the breakpoint types, we had the users retrospectively label the type of the breakpoints that were detected by Oasis.

Sixteen users participated in this phase of the evaluation, consisting of graduate and undergraduate students. A diagram editing task (design a leisure space for students) and a programming task (develop a set of digital image filters) were created and performed using MS Visio and Visual Studio. The tasks were designed to be open-ended, meaning that users had complete freedom in how they would achieve the goal; the task steps were not predetermined in any way. The tasks were more challenging and of longer duration than the controlled tasks typically used in prior lab studies. Eight users performed the diagram editing task while the other eight users performed the programming task. The tasks lasted about two hours.

During the study, users received two types of notifications: those relevant to the task and those of general interest to the user. For example, relevant notifications included references to useful examples of source code or leisure space diagrams whereas general interest notifications included actual references to new releases of common software or announcements related to our institution. Relevance was included as a factor to understand how it interacts with the system's scheduling behavior. Sixteen notifications were delivered per user, eight relevant and eight of general interest. An application was created

that generated notification requests at random moments during the user's task and rendered the notifications when the requests were granted.

Our system was configured to grant requests under two conditions. In one condition, requests were granted immediately and this served as a baseline. For the other condition, the system granted requests using the defer-to-next breakpoint policy. Users identified the types of the breakpoints retrospectively, allowing us to analyze the scheduling behavior for each type of breakpoint. Given the scope and complexity of the study, testing the other policies was not included here but should be a focus of future work. Dependent measures included the user's reaction time (time to acknowledge a notification), resumption time (the time to resume the main task), and self-reported frustration. We also gained user feedback about the effectiveness of the scheduling behavior through post experiment interviews.

A compelling finding from our study was that the concept of scheduling notifications at breakpoints fit well with how the users preferred the notifications to be managed. For example, when asked to characterize their preferred moments for a notification, many users described how they preferred, "finishing the current thing [action] rather than jumping to something else." More specifically, users pinpointed moments such as after adding specific shapes to a diagram or completing specific lines of source code, and describing how these actions related to the completion of their current goal; the same behavior our system seeks to mimic. Quantitatively, we found that scheduling notifications at breakpoints reduced frustration by 20% and reaction time by 25% relative to immediate delivery. This reduction in cost was balanced against notifications being deferred for only about 90 seconds on average.

User feedback also provided insights into when applications should specify each of the tested policies. For example, an application should choose a defer-to-medium or defer-to-fine policy for a notification relevant to the user's task and choose a defer-to-coarse policy for notifications of general interest. The rationale expressed by users is that they wanted to receive relevant notifications while they were still in the context of the main task, otherwise they felt the need to return to it; whereas they wanted the general interest notifications to be deferred until after they had left the main task and were more amenable to receiving general information. Overall, the results from our study show that scheduling notifications at breakpoints can result in a measurable benefit for users and fits their own expectations for managing notifications. However, the study also clearly indicates that more research is needed to improve breakpoint detection, preferably in a manner that does not impose a burden that may inhibit user adoption of this type of system in practice.

4.2 Constructing a Notification Request

When an application wants to render a notification using our system, it must construct and send a notification request. Each notification request consists of two components: the scheduling policy and a time window within which the notification must be delivered (see Section 3.1). The application must select the time window carefully because it will affect the system's scheduling

Table III.
Recommended values (in seconds) for the time window for each of the scheduling policies in Oasis

Defer-to-coarse	Defer-to-med	Defer-to-fine	Defer-to-any	Defer-to-best
190	158	114	225	659

behavior. For example, a time window that is set too short could override the corresponding policy because the likelihood of detecting a matching breakpoint would be low. On the other hand, setting the time window too long could reduce the utility of the notification's content, (e.g., if a matching breakpoint does not occur, the notification would be delayed until the time window expires). To assist application developers, we offer initial heuristics and discuss additional parameters that may be considered for specifying an appropriate time window.

An initial set of heuristics for appropriate time windows for each policy was derived by analyzing the breakpoint data set reported in Iqbal and Bailey [2007]. The heuristics are summarized in Table III. This dataset included the breakpoint distributions for several programming, document editing, and image manipulation tasks and was the most robust dataset we had available. Our goal in analyzing the dataset was to calculate durations for which there would be a reasonable likelihood of a breakpoint occurring that would satisfy the given policy. For example, for the defer-to-coarse, -medium, and -fine policies, we computed the heuristic based on the median distance between each of those types of breakpoints in the data set. For the defer-to-any policy, we used the median distance between all adjacent pairs of breakpoints. For the defer-to-best policy, we computed the heuristic as the median distance to the next breakpoint of any type and then added the maximum distance from any breakpoint to the next Coarse. The heuristics could be later refined by examining a larger data set and leveraging the experience gained from use of the system.

In selecting an appropriate time window as well as the scheduling policy, an application may want to consider parameters such as the urgency and relevance of the notification content with respect to the user's ongoing task. For example, if an application is aware that the notification is more urgent for the user or relevant to her task (e.g., using systems like Giornata [Voida and Mynatt 2009] or Tasktracer [Dragunov et al. 2005]), it may specify a shorter time window in its request (the exact value would require some calibration). This would allow opportunity for a suitable breakpoint to occur but help guard the utility of the notification. Though challenging, applications may leverage existing techniques for computing these values dynamically [Marx and Schmandt 1996; Horvitz et al. 1999] or, in the future, may be able to query system services to retrieve these values automatically [Dragunov et al. 2005]. One may also envision a system like Oasis making a user's current activities (the structure of which it already detects) available to applications, with some level of abstraction, along with predictions for how long s/he may remain on that activity as well as probabilities of switching to other activities. Applications may be able to leverage this information in determining how relevant or urgent their information is relative to the current task and create notification requests accordingly. Determining relevance or urgency is challenging, and the effect of inaccurate assessment is not negligible. Applications or services may therefore

consider establishing a user feedback mechanism to tune their algorithmic estimations of these attributes, which would help to improve the efficacy of the overall system behavior.

4.3 Granularities of Breakpoints

Our system schedules notifications by detecting interruptible moments during user tasks. As a result, the system is not able to provide measures of a user's interruptibility between these moments (i.e., it does not provide a continuous cost function). This capability would be useful, for example, for enabling additional scheduling policies in our system. One method for converging toward a more continuous cost function consistent with our current approach is to detect successively finer granularities of breakpoints (e.g., in the extreme case, down to the transitions between keystrokes). However, we have not pursued this level of detail in our system because the empirical evidence has not yet shown that delivering notifications at breakpoints with a granularity finer than those already detectable by our system yields a measurable benefit for the user [Iqbal and Bailey 2006]. That is, delivering a notification at a breakpoint with a very fine granularity has not been shown to be better than delivering it immediately.

Conversely, it might also be useful to detect additional granularities of breakpoints coarser than those currently detectable by our system. An example of this type of breakpoint would be a transition between physical activities within the office (e.g., switching between interacting with the desktop and reading physical documents). This type of breakpoint could be detected by deploying sensors throughout the office environment [Fogarty and Hudson 2007] and learning statistical models using a process and tools similar to those described in this article. Because empirical benefits of using this class of breakpoint have been shown in the domain of mobile computing [Ho and Intille 2005], notification management systems similar to Oasis should consider implementing this extension.

4.4 Generalizing to Other Tasks, Domains, and System Functionalities

Oasis demonstrates the technique of aligning notification scheduling to the perceptual structure of user tasks and the system was implemented in the desktop domain as a proof of concept. An evaluation offered initial evidence indicating that the system provides an empirical benefit for users during authentic activities and that the system's behavior is consistent with how users prefer notifications to be managed. The evaluation tested the system in the two task domains that it currently supports: programming and diagram editing. In addition, prior work has tested the technique and shown favorable results for other interactive tasks, including document editing, image manipulation, and communication tasks. We therefore believe the technique generalizes to the class of goal-directed, content generation tasks. Further research is needed to test the efficacy of our system for interactive tasks that are less goal-directed or do not involve content generation. For example, one challenge with these types of tasks is that the perceptual structure may be more difficult to detect, by both systems and human observers. For example, if a user is browsing the Web for

leisure (where the goal is less obvious and there is no content generation), it may be more difficult to reliably identify the transitions between meaningful units of action.

Extending the capability of a system like Oasis to other desktop task domains would require that the event streams of the desired applications be made available to the system. As application plug-in and scripting architectures have become common, the technical feasibility for exposing the event streams is already available. For example, we were able to create plug-ins for MS Visio and Visual Studio with relatively modest effort. And in the TaskTracer project [Dragunov et al. 2005], it was demonstrated that plug-ins could be created to capture a very large number of interaction events for the entire MS Office suite (Excel, Word, Powerpoint, and Outlook). Generally speaking, developers should report as many events as possible and allow the model building component to learn which events to include in the breakpoint models. In addition, the effect on the user experience caused by a growing number of notifications in the desktop domain provides a strong incentive for developers to create these types of plug-ins. For example, an application that interacts with a notification management system for delivering frequent information updates would have a clear user experience and marketing advantage over competing applications that do not provide similar support.

Though our system was implemented in the desktop domain, the basic technique of deferring notifications until breakpoints can be effective in other domains where users perform goal-directed tasks. For example, for mobile computing, researchers have already found that deferring notifications until the user initiates a physical transition (e.g., from sitting to standing) yields positive results relative to immediate delivery [Ho and Intille 2005]. In the domain of ubiquitous computing, researchers have found that transitions between different task and leisure activities within the home would also serve an important role in notification management [Nagel et al. 2004]. In safety critical domains such as driving, notifications from in-vehicle information systems could be deferred until the driver completes a critical action such as turning, stopping at a light, or reaching a steady speed. However, in this environment, systems would likely need to consider other contextual factors such as weather and traffic conditions in addition to the sequence of localized driver actions when deciding when to render notifications.

Finally, system functionalities beyond notification management could be enabled with access to the perceptual structure of user tasks. For example, this information could be used by content generation tools (e.g., programming, image manipulation, and document editing tools) to organize a linear sequence of user commands into a hierarchical structure. For example, this structure would enable users to perform undo and redo actions at different levels of granularity or could be used to create a better visual summary of the command histories.

5. CONCLUSIONS AND FUTURE WORK

Users rely on notifications to maintain awareness of desired information. Because the applications or services delivering notifications have no capability of

understanding the state of the user's task, they often render their notifications at overly disruptive moments. Empirical research has shown favorable results for using perceptual breakpoints during task execution as less disruptive moments for notification, but it has been unclear whether this strategy could be effectively implemented or what the effects on end users would be for authentic activities.

This article has contributed the design of a system called Oasis that demonstrates the feasibility of aligning notification delivery with the perceptual structure of user tasks. The key aspects of our system are that it detects task structure without any explicit knowledge of the task itself, layers flexible scheduling policies on top of the detection mechanism, and provides a service that applications can access for scheduling notifications with these policies. An evaluation showed that the system can provide empirical benefits for the user and led to initial recommendations for how and when each of the policies is best utilized. Our system has been fully implemented and other researchers can use it to further explore notification scheduling policies and other system functionalities based on the use of task structure. The source code of Oasis can be obtained by contacting S. T. Iqbal.

We see several directions for future work. One immediate direction is to explore how to build more robust default statistical models for breakpoint detection or methods for creating personalized models that limit the training overhead for users. Another direction is to conduct longer-term studies of how a notification scheduling system like Oasis affects users and their tasks and study the use of the system for a more diverse task set. A third direction is to better understand if and how users develop a mental model of the system's behavior and how they react to this type of automated decision making. A fourth direction is to explore development of additional notification scheduling policies that leverage task structure and other contextual factors. Finally, it may be fruitful to explore how the use of perceptual task structure could enable or improve other types of system functionalities, such as automatically creating structured command histories.

ACKNOWLEDGMENTS

We thank the users who volunteered to participate in our related studies.

REFERENCES

Adamczyk, P. D. and Bailey, B. P. 2004. If not now when? The effects of interruptions at different moments within task execution. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 271–278.

Bailey, B. P., Adamczyk, P. D., Chang, T. Y., and Chilson, N. A. 2006. A framework for specifying and monitoring user tasks. *J. Comput. Hum. Behav. 22*, 4, 685–708.

Bailey, B. P. and Iqbal, S. T. 2008. Understanding changes in mental workload during execution of goal-directed tasks and its application for interruption management. *ACM Trans. Comput. Hum. Interac. 14*, 4, 1–28.

Bailey, B. P. and Konstan, J. A. 2006. On the need for attention aware systems: measuring effects of interruption on task performance, error rate, and affective state. *J. Comput. Hum. Behav. 22*, 4, 709–732.

BEGOLE, J. B., MATSAKIS, N. E., AND TANG, J. C. 2004. Lilsys: sensing unavailability. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. 511–514.

CZERWINSKI, M., CUTRELL, E., AND HORVITZ, E. 2000a. Instant messaging and interruption: influence of task type on performance. In *Proceedings of the Annual Conference of the Human Factors and Ergonomics Society of Australia (OZCHI)*. C. Paris, N. Ozkan, S. Howard, and S. Lu, Eds., 356–361.

CZERWINSKI, M., CUTRELL, E., AND HORVITZ, E. 2000b. Instant messaging: effects of relevance and timing. In *Proceedings of HCI: People and Computers XIV*. S. Turner and P. Turner, Eds., British Computer Society, 71–76.

CZERWINSKI, M. AND HORVITZ, E. 2002. Memory for daily computing events. In *Proceedings of HCI: People and Computers XVI*. F. Culwin, Ed.

DABBISH, L. AND KRAUT, R. E. 2004. Controlling interruptions: awareness displays and social motivation for coordination. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. 182–191.

DEMARCO, T. AND LISTER, T. 1999. *Peopleware: Productive Projects and Teams* 2nd Ed. Dorset House Publishing Company, New York.

DEY, A. K. AND ABOWD, G. D. 2000. CybreMinder: a context-aware system for supporting reminders. In *Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing*. 172–186.

DISMUKES, K., YOUNG, G., AND SUMWALT, R. 1998. Cockpit interruptions and distractions. *ASRS Directline 10*.

DRAGUNOV, A. N., DIETTERICH, T. G., JOHNSRUDE, K., MCLAUGHLIN, M., LI, L., AND HERLOCKER, J. L. 2005. TaskTracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the International Conference on Intelligent User Interfaces*. 75–82.

FOGARTY, J. AND HUDSON, S. E. 2007. Toolkit support for developing and deploying sensor-based statistical models of human situations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, 135–144

FOGARTY, J., KO, A. J., AUNG, H. H., GOLDEN, E., TANG, K. P., AND HUDSON, S. E. 2005. Examining task engagement in sensor-based statistical models of human interruptibility. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 331–340.

FOGARTY, J., LAI, J., AND CHRISTENSEN, J. 2004. Presence versus availability: The design and evaluation of a context-aware communication client. *Int. J. Hum. Comp. Studies, 61*, 3, 299–317.

GILLIE, T. AND BROADBENT, D. 1989. What makes interruptions disruptive? A study of length, similarity, and complexity. *Psych. Rese. 50*, 243–250.

GLUCK, J., BUNT, A., AND MCGRENERE, J. 2007. Matching attentional draw with utility in interruption. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM Press, 41–50

GONZALEZ, V. M. AND MARK, G. 2004. "Constant, constant, multi-tasking craziness": managing multiple working spheres. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 113–120.

HO, J. AND INTILLE, S. S. 2005. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. ACM Press, 909–918

HORVITZ, E. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 159–166.

HORVITZ, E., APACIBLE, J., AND SUBRAMANI, M. 2005. Balancing awareness and interruption: investigation of notification deferral policies. In *Proceedings of the 10th User Modeling International Conference*. L. Ardissono, P. Brna, and A. Mitrovic, Eds., Springer, 433–437.

HORVITZ, E., JACOBS, A., AND HOVEL, D. 1999. Attention-sensitive alerting. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. 305–313.

HORVITZ, E., KADIE, C. M., PAEK, T., AND HOVEL, D. 2003. Models of attention in computing and communications: from principles to applications. *Comm. ACM*, 52–59.

HORVITZ, E., KOCH, P., KADIE, C. M., AND JACOBS, A. 2002. Coordinate: probabilistic forecasting of presence and availability. In *Proceedings of the 18th Conference on Uncertainty and Artificial Intelligence*. Morgan-Kaufmann, 224–233.

IQBAL, S. T. 2008. A framework for intelligent notification management in multitasking domains. Ph.D. Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign.

IQBAL, S. T. AND BAILEY, B. P. 2005. Investigating the effectiveness of mental workload as a predictor of opportune moments for interruption. In *Proceedings of the ACM Conference on Human Factors in Computing Systems.* 1489–1492.

IQBAL, S. T. AND BAILEY, B. P. 2006. Leveraging characteristics of task structure to predict costs of interruption. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 741–750.

IQBAL, S. T. AND BAILEY, B. P. 2007. Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 697–706.

IQBAL, S. T. AND BAILEY, B. P. 2008. Effects of intelligent notification management on users and their tasks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 93–102.

IQBAL, S. T. AND HORVITZ, E. 2007. Disruption and recovery of computing tasks: field study, analysis and directions. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 677–686.

IQBAL, S. T. AND HORVITZ, E. 2010. Notification and awareness: a field study of alert usage and preferences. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW).* ACM, 27–30.

JACKSON, T. W., DAWSON, R. J., AND WILSON, D. 2001. The cost of email interruption. *J. Syst. Inform. Technol. 5*, 1, 81–92.

KREIFELDT, J. G. AND MCCARTHY, M. E. 1981. Interruption as a test of the user-computer interface. In *Proceedings of the 17th Annual Conference on Manual Control*. Jet Propulsion Laboratory, California Institute of Technology, JPL Publication 81–95, 655–667.

LATORELLA, K. A. 1998. Effects of modality on interrupted flight deck performance: implications for data link. In *Proceedings of the 42nd Annual Meeting of the Human Factors and Ergonomics Society*. 87–91.

LATORELLA, K. A. 1999. Investigating interruptions: Implications for flightdeck performance. National Aviation and Space Administration, Washington, DC.

LEE, J. D., HOFFMAN, J. D., AND HAYES, E. 2004. Collision warning design to mitigate driver distraction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 65–72.

MAES, P. 1994. Agents that reduce work and information overload. *Comm. ACM 37*, 7, 30–40.

MAGLIO, P. AND CAMPBELL, C. S. 2000. Tradeoffs in displaying peripheral information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*. 241–248.

MARX, M. AND SCHMANDT, C. 1996. CLUES: Dynmic prsonalized message filtering. In *Proceedings of the ACM Conference on Computer Supported Cooperative work*.

MCCRICKARD, D. S., CATRAMBONE, R., CHEWAR, C. M., AND STASKO, J. T. 2003. Establishing tradeoffs that leverage attention for utility: Empirically evaluating information display in notification systems. *Int. J. Hum.-Comput. Studies 58*, 5, 547–582.

MCFARLANE, D. C. 1999. Coordinating the interruption of people in human-computer interaction. In *Proceedings of the IFIP TC.13 International Conference on Human-Computer Interaction*. 295–303.

MCFARLANE, D. C. AND LATORELLA, K. A. 2002. The scope and importance of human interruption in HCI design. *Hum.-Comput. Interact. 17*, 1, 1–61.

MIYATA, Y. AND NORMAN, D. A. 1986. Psychological issues in support of multiple activities. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, D. A. Norman and S. W. Draper Eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 265–284.

MONK, C. A., BOEHM-DAVIS, D. A., AND TRAFTON, J. G. 2002. The attentional costs of interrupting task performance at various stages. In *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting*.

NAGEL, K. S., HUDSON, J. M., AND ABOWD, G. D. 2004. Predictors of availability in home life context-mediated communication. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. ACM Press, 497–506.

NAVON, D. AND GOPHER, D. 1979. On the economy of the human processing system: a model of multiple capacity. *Psych. Rev. 86*, 254–255.

NEWTSON, D. 1973. Attribution and the unit of perception of ongoing behavior. *J. Person. Soc. Psych. 28*, 1, 28-38.

NEWTSON, D. AND ENGQUIST, G. 1976. The perceptual organization of ongoing behavior. *J. Experi. Soc. Psych. 12*, 436–450.

SPEIER, C., VALACICH, J. S., AND VESSEY, I. 1999. The influence of task interruption on individual decision making: An information overload perspective. *Decis. Sci. 30*, 2, 337–360.

STANTON, N. 1994. *Human Factors in Alarm Design.* Taylor and Francis, London, UK.

TANG, J. C., YANKELOVICH, N., BEGOLE, J., KLEEK, M. V., LI, F., AND BHALODIA, J. 2001. ConNexus to awarenex: Extending awareness to mobile users. In *Proceedings of the ACM Conference on Human Factors in Computing Systems.* ACM Press, 221–228.

VAN DANTZICH, M., ROBBINS, D., HORVITZ, E., AND CZERWINSKI, M. 2002. Scope: Providing awareness of multiple notifications at a glance. In *Proceedings of the Conference on Advanced Visual Interfaces*.

VOIDA, S. AND MYNATT, E. D. 2009. It feels better than filing: everyday work experiences in an activity-based computing system. In *Proceedings of the 27th International Conference on Human factors in Computing Systems.* ACM.

WICKENS, C. D. 2002. Multiple resources and performance prediction. *Theo. Issues Ergon. Sci. 3*, 2, 159–177.

WITTEN, I. H. AND FRANK, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA.

ZACKS, J., TVERSKY, B., AND IYER, G. 2001. Perceiving, remembering, and communicating structure in events. *J. Exper. Psych. Gen. 130*, 1, 29–58.

ZACKS, J. M. AND TVERSKY, B. 2001. Event structure in perception and conception. *Psych. Bull. 127*, 3–21.

ZIJLSTRA, F. R. H., ROE, R. A., LEONORA, A. B., AND KREDIET, I. 1999. Temporal factors in mental work: effects of interrupted activities. *J. Occup. Organiz. Psych. 72*, 163–185.

Received June 2009; revised March 2010, July 2010; accepted August 2010