Effects of Intelligent Notification Management on Users and Their Tasks

Shamsi T. Iqbal and Brian P. Bailey
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
{siqbal, bpbailey}@uiuc.edu

ABSTRACT
We present a novel system for notification management and report results from two studies testing its performance and impact. The system uses statistical models to realize defer-to-breakpoint policies for managing notifications. The first study tested how well the models detect three types of breakpoints within novel task sequences. Results show that the models detect breakpoints reasonably well, but struggle to differentiate their type. Our second study explored effects of managing notifications with our system on users and their tasks. Results showed that scheduling notifications at breakpoints reduces frustration and reaction time relative to delivering them immediately. We also found that the relevance of notification content determines the type of breakpoint at which it should be delivered. The core concept of scheduling notifications at breakpoints fits well with how users prefer notifications to be managed. This indicates that users would likely adopt the use of notification management systems in practice.

Author Keywords
Breakpoint, Interruption, Notification, Statistical models

ACM Classification Keywords
H.1.2 [Models and Principles]: User/Machine Systems – human information processing and human factors

INTRODUCTION
Task interruptions caused by notifications in the desktop can negatively impact users, e.g., increase task time and frustration [1, 3]. This is a significant problem as such notifications occur many times during the work day [23].

A promising solution to this problem is to effectively mediate the delivery of notifications [24]. Two threads of research have strived to realize this solution. One thread has probed strategies for detecting interruptible moments leveraging the structure of users' tasks. For example, scheduling notifications at *breakpoints* has been shown to reduce cost of interruption [1, 3, 21]. But, this strategy has only been tested for controlled tasks, and it is not clear whether similar positive results would occur for authentic, free-form tasks. The other thread has focused on building statistical models to detect interruptible moments during authentic tasks [8, 16, 22], but the effects of scheduling notifications using these models has not been studied.

This paper merges and extends both of these threads. First, we present a new system that realizes defer-to-breakpoint policies for managing notifications during authentic tasks. The system utilizes statistical models for detecting breakpoints during these tasks. Plug-ins for applications were developed to study the performance and impact of the system in context of two complex task domains, diagram editing and programming.

Second, we evaluate how well statistical models detect three granularities (types) of breakpoints within *novel task sequences*. By novel task sequences, we mean those that were not used as part of the model building process. This is important since the overall effectiveness of our system hinges on the models' performance in practice. For this work, we use *composite* models, built by aggregating training data from multiple users [22]. We focus on composite models since they can serve as default models in our system, but their effectiveness is not yet known. We seek to differentiate breakpoints since three types of breakpoints have been shown to exist in user tasks [22], and detecting them allows for more flexible scheduling. Our results show that the models can detect breakpoints reasonably well (55.5% on average), but struggle to differentiate their type. The main implication is that these models are most useful for *detecting* breakpoints without differentiating type in notification management systems.

Finally, we report results from a second user study that tested how scheduling notifications at different types of breakpoints impacts users and their tasks relative to delivering them immediately (today's UI). Breakpoints were being detected by our system in real time using the models from the first study. To compensate for the results of our first study, we had users identify the type of each breakpoint detected retrospectively. Results show that scheduling notifications at breakpoints reduces users' frustration and reaction time relative to immediate. This was balanced against a deferral time of 1.5 minutes on

CHI 2008, April 5–10, 2008, Florence, Italy.

average. The relevance of the notification's content to the ongoing task was found to govern the type of breakpoint at which it should be scheduled. The overall concept of scheduling notifications at breakpoints fit well with how users themselves preferred notifications to be managed. This strongly suggests that users would indeed be willing to adopt notification management systems in practice.

RELATED WORK

We describe breakpoints, their use for notification management, and how they can be detected. We discuss approaches for predicting other interruptible moments and how the predictions can be used to schedule notifications.

Breakpoints, Their Use, and How to Detect Them

A breakpoint represents the moment of transition between two observable, meaningful units of task execution [26], and reflects transitions in perception or action [27]. Prior work identified three granularities (types) of perceptually meaningful breakpoints during interactive tasks – Coarse, Medium, and Fine [22]. Coarse exist between the largest units while Fine exist between the smallest. Studies have shown that delivering notifications at breakpoints reduces interruption cost (resumption lag, frustration, errors), and coarser breakpoints correspond to lower cost [1, 3, 21].

A challenge is to detect and differentiate breakpoints in user tasks. One approach is to use statistical models that map user interaction to each type of breakpoint. Prior work has demonstrated that such models can detect and differentiate breakpoints with 69-87% accuracy [22]. These accuracies were reported by testing the models on the *original* training data using 10-fold cross validation.

Our work investigates how well composite models detect and differentiate breakpoints in *novel* task sequences. This means that their performance is being evaluated on data generated from new users performing new tasks in the same domain. This will provide understanding of how well such models would enable notification management systems to implement defer-to-breakpoint policies.

Detecting Other Interruptible Moments

Other work has developed statistical models for detecting interruptible moments, but not breakpoints explicitly. These models typically leverage cues related to desktop activity, visual and acoustical analysis of the physical environment, and scheduled user actions [13, 15, 17, 19]. For example, Horvitz and Apacible use these cues to infer a probability distribution over users' attentional state, from which interruption costs are computed [13]. Fogarty et al. built statistical models that map interaction events (typing, scrolling, browsing, etc.) to one of three classes of task engagement [8], where reaction time to a secondary task was used as ground truth. While models in this corpus of work have been shown to effectively predict interruptibility, they have either not been utilized for scheduling notifications or the effects of scheduling notifications have not been studied for authentic tasks.

Beyond this body of work, our research further studies the use of statistical models for detecting and differentiating breakpoints in complex tasks. Using these models, we also investigate the effects of scheduling notifications at breakpoints on users and their tasks.

Scheduling Notifications at Interruptible Moments

Only a few systems exist that schedule or are capable of scheduling notifications. For example, the Lookout system predicts a user's dwell time on a communication message based on an analysis of its content [12]. This prediction is then used to schedule delivery of automated assistance. Another system, the Notification Platform, modulates the flow of messages from multiple sources to devices by performing ongoing decision analysis [17]. In this system, messages are delivered using the device and modality most beneficial for the user. A related system, BESTCOM, considers social and task context, available channels, and communication preferences to select the best timing and modality for interpersonal communication [18]. Other similar systems have also been created [4, 9].

Relative to this work, a contribution of our research is that we studied how automated scheduling of notifications affects users and their tasks. We studied one technique for scheduling, deferring notifications until breakpoints. But our results may help understand the effects of notification scheduling using other systems and improve their design.

OASIS: A NOTIFICATION MANAGEMENT SYSTEM

OASIS is a system that allows notifications to be deferred until breakpoints are reached during interactive tasks. This approach allows notifications to be presented in a timely manner, but at moments that have been shown to correspond with reduced interruption cost [1, 3, 11, 21].

As illustrated in Figure 1, OASIS consists of two primary components; the breakpoint detector and the scheduler. When an application wants to render a notification, it sends a request to the scheduler. The request consists of a flag indicating which policy to use and a maximum time it can wait. As a first step, four policies are supported; defer until next *fine*, *medium*, or *coarse* breakpoint; and defer until next breakpoint of *any* type. The scheduler queues the request until the specified type of breakpoint occurs or until the timeframe expires. In either case, the request is granted and the application can render its notification. For example, if a user is constructing a diagram and an e-mail notification is generated, our system would allow it to be deferred until the user finishes their current manipulation of shapes (Fine), saves the current diagram (Medium), or switches to a completely different activity (Coarse).

The breakpoint detector monitors the user event stream. Events are pooled for a few seconds and are then fed into a set of statistical models. The models, organized similar to those reported in [22], detect which type of breakpoint occurred, if any. The result is passed to the scheduler, which compares it to the policy specified in the request

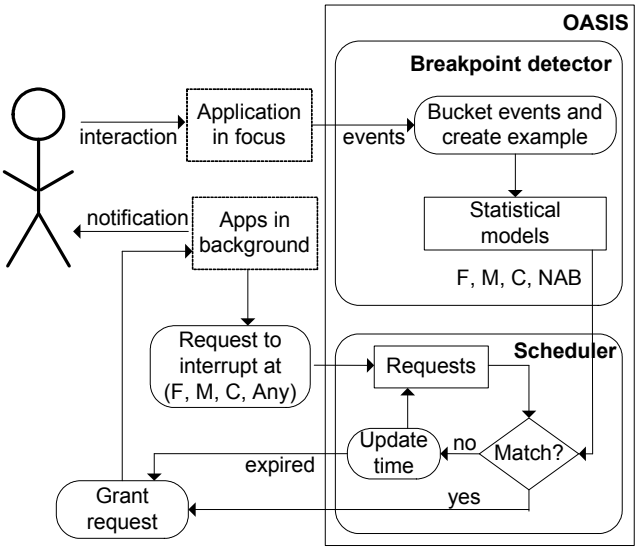

Figure 1. A schematic of OASIS, our notification management system. When an application wants to render a notification, it sends a request to OASIS. OASIS monitors the event stream and defers the request until an appropriate breakpoint is detected or the specified timeframe expires. Requests can be deferred until the next Fine (F), Medium (M), or Coarse (C) breakpoint, or until the next breakpoint of any type (Any).

and then takes the appropriate action. OASIS provides an offline component for training the models, but the models could also be trained using other toolkits, e.g., Subtle [6].

EVALUATION OVERVIEW

Using OASIS, we wanted to answer two main research questions about notification management:

- How well can a notification management system detect and differentiate breakpoints within novel task sequences, i.e., sequences that lie outside of the data set originally used for training the statistical models?
- How does scheduling notifications at breakpoints affect users and their tasks? For example, does this reduce frustration or resumption lag, and how do users react to this type of scheduling behavior?

The first question is critical because to understand effects of scheduling notifications at breakpoints during authentic tasks, we need to understand how well the system can detect breakpoints. This is an important challenge since users' task behavior is known to be *highly* variable. It is thus not clear how well models for detecting breakpoints trained with one data set would perform on a different set.

The second question is important because effects of scheduling notifications have only been explored using controlled and relatively simple tasks where the moments were selected a priori (e.g., [1, 2, 5, 20, 21, 25]). It is thus unknown whether the previous findings generalize when notifications are scheduled during authentic, complex tasks and when the breakpoints are identified in real time.

Task Domains

Two domains were selected for this work; programming and diagram editing. These were selected because many users perform tasks in these domains, the tasks performed are typically complex, and the tasks are often intertwined with other activities such as Web browsing or managing communications. Microsoft's Visual Studio and Visio were chosen as the specific applications for programming and diagram editing, respectively. For each application, we developed custom plug-ins that expose a large number of application events that can be monitored by our system.

Training Initial Models

Following procedures outlined in [22], we trained a set of statistical models for detecting breakpoints when working within Visual Studio and Visio (Fine and Medium), and when switching between higher-level activities (Coarse).

Six users (3 per domain) were recruited and our data collection software was installed on their machines. Users were asked to run the software the next time they would be focused on performing any task within the assigned domain for at least 90 minutes. They were also asked to perform the activity as usual, i.e., it was perfectly fine to check mail, play music, or read news intermittingly. Our software collected user's screen interaction, application and system-level events, and keyboard and mouse events.

The users performed a diverse set of complex tasks. For programming, one user was working on a Web-based graphics application using ASP.net. The second user was programming a notification display using Visual C#. The third user was writing C++ code to manipulate mouse and keyboard events for a distributed application. For diagram editing, one user was creating an information architecture for a research website. The second user was creating a project proposal outline for the local environment council. The third user was creating a system diagram for a poster.

Twelve independent observers were then recruited. Using a software tool, each observer reviewed two interaction videos and identified locations of perceived breakpoints and their type (fine, medium, coarse). The aggregated set of identified breakpoints was filtered to include only those breakpoints for which there was a minimum threshold of agreement. Details of this process can be found in [22].

For moments identified as breakpoints, training examples were created from events surrounding each breakpoint. All other moments were coded as training examples corresponding to NABs (*not* a breakpoint). This yielded the most comprehensive models possible with our data. Training examples included a wide range of events, achieved by deriving some feature values (e.g. changes in cursor direction) from other events (e.g. mouse position events within a given time interval). Examples were aggregated across users within each task domain to create composite statistical models, as analogous work has

		Predicted	
		Coarse	Not Coarse
Actual	Coarse	60	25
	Not Coarse	19	2188

Table 1. Predicted vs. Actual for the general model used to detect Coarse breakpoints. Overall accuracy was 97.97%.

		Predicted		
		Med	Fine	NAB
Actual	Med	64, 40	0, 12	12, 20
	Fine	0, 0	104, 93	4, 67
	NAB	16, 5	9, 16	1034, 711

Table 2. Predicted vs. Actual for the application models used to detect Medium and Fine for (programming, diagram editing). Overall accuracies were 96.7% and 87.6%, respectively.

shown this to result in good performance [13]. Models were learned using the Weka machine learning toolkit.

The general model for detecting Coarse had 11 features (predictive events), while application specific models for Medium and Fine within Visual Studio and Visio had 13 and 17, respectively. For Coarse, features included *switch to mail client*, *switch to IM client*, and *window minimized*. For Medium and Fine in Visual Studio, features included *document closing*, *build done*, and *switch to search*. For Visio, features included *shape added, application deactivated, document saved*, and *begin zooming*.

Training Results

Model performances are shown in Tables 1 and 2, with correct predictions shown along the diagonals. Accuracies (percentage of correct predictions) were high (> 87%). This is due to the large number of NABs and the models having predicted most of them correctly (general model: 99%, programming: 98%, and diagram editing: 97%).

To evaluate how well the models predict actual breakpoints, we computed the Recall values (percent of actual cases correctly predicted) for each breakpoint type. Recall of Coarse was 71% (60/85). Recall of Medium was 84% for programming and 56% for diagram editing. Recall of Fine was 96% for programming and 58% for diagram editing. The models missed some breakpoints. This means that occasionally opportunities for scheduling notifications at breakpoints would be missed. However, the most egregious error, wrongly predicting a moment to be a breakpoint when it is not, was very low (<0.5%).

Recall values showed that our models are able to detect and differentiate breakpoints in the training data with reasonable accuracy. Results were consistent with prior work [22]. This gave us high confidence that these models were robust enough for testing on novel task sequences.

STUDY 1: EVALUATE BREAKPOINT DETECTION

The purpose of the first study was to evaluate how well breakpoints could be detected and differentiated within novel task sequences, i.e., sequences generated outside the data originally used for training and evaluation.

Users and Tasks

Six users (3 per domain) were recruited. Each user reported having moderate to expert skills in the respective domain. We installed our data collection software on the user's machines, allowing them to work on their own tasks and in their own environments. Users were asked to run our software the next time that they would be focused on performing their task (with the assigned application) for an hour or more. Users were informed that they should maintain normal work practices, e.g. switch applications, chat with others, or browse the Web as desired.

For programming, one user was developing a graphical interface for a mobile device using Visual C#. Another user was programming a graphics rendering tool using Visual C++. The third user was writing code to process image files in Visual C#. For diagram editing, one user was outlining her doctoral thesis. The second user was diagramming the logic flow of an interactive game. The third was creating a process diagram for a research paper.

Procedure

The procedure consisted of two phases. In the first phase, users worked on their selected tasks with our software running. Our system monitored the event stream and used the originally trained models to detect whether and what type of breakpoint had occurred. Event data was pooled in 3 second bins, as this was found in the training phase to give the best accuracy. Each bin of event data and the related prediction (Fine, Medium, Coarse, or NAB) were logged to a file. Users' screen interaction was recorded, and could be synchronized with the event data.

For the second phase, users used a software tool to review their own interaction videos and identify locations of the breakpoints and their type. We asked the users themselves to identify the breakpoints, rather than utilize independent observers, because a system like ours will ultimately be evaluated based on how well its predictions match a user's own understanding of their tasks. Note that using observers in the training phase was important because it allows the most perceptually salient breakpoints to be detected and used for training, resulting in robust models.

Measurements and Analysis

We compared the breakpoints detected by the composite models to the breakpoints identified by the users. A system-identified breakpoint was considered a match with a user-identified breakpoint if they were within 10s and of the same type. After testing several values, a 10s window seemed to best compensate for the difference between when a breakpoint occurred and when a user annotated it.

Results

Tables 3 and 4 show the distribution of system- vs. user-identified breakpoints, with the diagonals showing the

		System-identified			
		Coarse	Medium	Fine	NAB
User-identified	Coarse	44	17	2	43
	Medium	40	21	1	41
	Fine	35	17	18	49
	NAB	69	12	8	3092

Table 3. System- vs. User-identified breakpoints for programming. Overall accuracy was 90.5%.

		System-identified			
		Coarse	Medium	Fine	NAB
User-identified	Coarse	33	12	0	35
	Medium	22	5	1	22
	Fine	25	1	1	29
	NAB	68	7	0	3299

Table 4. System- vs. User-identified breakpoints for diagram editing. Overall accuracy was 93.8%.

matches. Data was aggregated across users in each domain. The overall accuracies were high (> 90%), but again, this is due in part to the large number of NABs, and the ability of the models to correctly predict most of them.

Recall of Coarse was 41.5% and 41.3% for programming and diagramming, respectively. Recall of Medium was 20.4% for programming and 10% for diagram editing. Recall of Fine was 15% for programming and only 1.7% for diagram editing. Admittedly, these results were much lower than expected given the training performance.

Closer inspection revealed that the majority of mismatches were due to users identifying breakpoints as Medium or Fine, while the system identified those same breakpoints as Coarse. On the one hand, this is in fact a promising result because it shows that users and the system were agreeing on the *location* of the breakpoints, but were disagreeing on the *type* of those breakpoints. Part of the reason behind the low accuracy in identifying the type of breakpoint was the inability of the models to understand users' task context. For example, one user switched repeatedly between Gmail and Visio to retrieve documents related to her task. The system identified these switches as Coarse breakpoints, while the user identified them as Medium or Fine considering the relevance to the ongoing task. This illustrates how task context influences perceptions of breakpoint type as well as the necessity and challenge of integrating such context into the models. The most egregious type of error, detecting a breakpoint when none was present, was still very low; 2.8% for programming and 2.3% for diagramming.

Discussion

Results from this study highlight the significant challenge of using composite models to detect and differentiate breakpoints within novel task sequences. Several methods could be pursued to increase the accuracy of such models. For example, models could be trained using a much larger data set, they could be trained on a per user basis, or a combined approach could be followed. Including features representing additional task context must also be pursued.

Yet even if this problem can be mostly solved, and we believe that it will given the active ongoing research in this direction [6, 17, 18], a critical question still remains. Would scheduling notifications at breakpoints (assuming correct identification) have a positive impact for users?

To provide a first answer to this question, we wanted to build upon the fact that the models could identify the location of breakpoints with reasonable accuracy. We thus collapsed the breakpoints into one type and retrained our models to identify each moment as either a breakpoint or NAB. These new models resulted in 59% and 52% recall for programming and diagram editing. We judged this to be sufficient to move forward with our second study.

As part of the study, we also wanted to test the effects of scheduling notifications at each type of breakpoint. We decided to use our system (with the new models) to *detect* the locations of breakpoints and to ask the users to identify type. This would effectively compensate for the differentiation performance of the models, and allow breakpoint type to be included in the analysis. We felt that this was important because it would show whether the ability to differentiate breakpoint type for scheduling notifications would have any benefit for users in practice.

STUDY 2: EVALUATE EFFECTS OF SCHEDULING

The purpose of the second study was to evaluate how scheduling notifications at breakpoints impacts users and their tasks. We also wanted to investigate the interaction between notification content and scheduling policy. We thus designed notifications to be either relevant to the task or of general interest to the user (but not task relevant).

Users and Tasks

16 users (8 per domain) were recruited for the study. Users reported having moderate to expert skills in the respective domain. Users received $50 for participating.

Tasks required users to develop solutions to challenging, ill-structured problems during the study. Only high-level descriptions were provided, and it was up to the users to work out a desired solution. For programming, the task was to create a user interface for applying convolution filters to images, and to implement at least three filters. Users were provided with a description of convolution filters, pointers to Web-based resources, and a skeleton C# project that they could build upon, if desired. Users were asked to make their code as efficient and readable as possible. MS Visual Studio was used for the task.

For diagram editing, the task was to design a floor plan for a model work space in a Computer Science building. The space had to accommodate 6 students, and needed to include cubicles for daily work, a joint lab for conducting experiments, and a service room for relaxing and eating.

Users were asked to create as many design alternatives as possible. The task was performed using Visio.

For both tasks, the goal was to have users work in a manner similar to how they would in practice. Users were free to browse for examples, references, or any other desired information online. They were informed that they needed to spend 2 hours on the task and should work at their own pace. Given the length of the task, users were also free to perform other personal tasks such as check mail or read their favorite online news site. To facilitate motivation, an additional $50 was offered to the user who created the highest quality solution in each task domain. OASIS was installed on the experimental machine and monitored the user's activity to detect breakpoints. It also managed notification requests from a custom application.

Notifications

As users performed tasks, they occasionally received notifications. Two types of notifications were used:

- *Relevant*. These notifications provided examples (e.g., source code or floor plans), useful tips, or additional criteria that would be used for judging solution quality.
- *General Interest*. These notifications presented recent news grabbed from Google News or announcements from our department's or institution's homepage.

Two policies were used for delivering notifications:

- *Defer to breakpoint*. A request would be sent to OASIS, which would schedule the notification to appear at the next breakpoint detected in the user's task sequence.
- *Immediate*. The notification was delivered immediately.

16 notifications were generated randomly within intervals spanning the 2 hour task period by a custom application. 16 was based on prior work showing that computer users receive on average about eight notifications per hour [23]. 4 notifications appeared as soon as they were generated (Immediate). The other 12 were to be scheduled by OASIS to appear at breakpoints. However, depending on our system's detection of breakpoints, it was possible for users to receive fewer. We chose to have more scheduled (12) than immediate (4) notifications to try to ensure that each type of breakpoint would be used (how types were identified is discussed below). Half of the notifications were Relevant and the other half were of General Interest. These were balanced between the two scheduling policies.

Notifications appeared as a non-modal window in the lower right of the screen and contained a short snippet of the message (fig. 2). Users selected the snippet to read the full message. The design simulated commonly used alerting techniques. The window persisted for 7s.

Experimental Design

We used a 2 Activity (programming, diagram editing) X 2 Policy (breakpoint, immediate) X 2 Content (relevant, general interest) mixed design. Policy and Content were within subjects while Activity was between subjects.

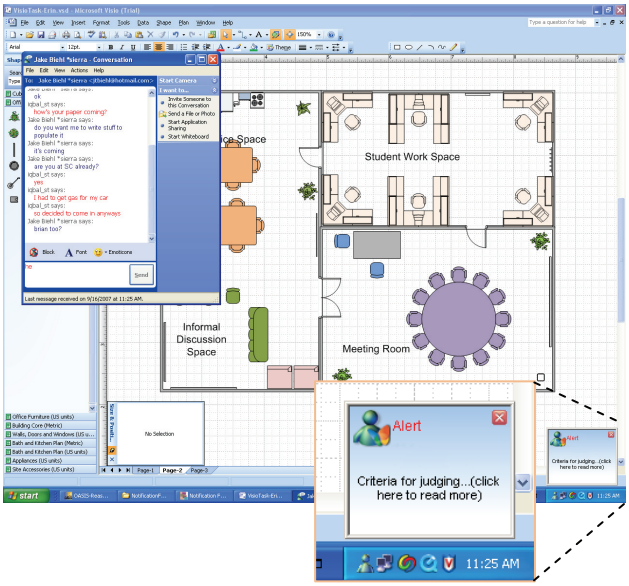

Figure 2. Screenshot of a notification arriving as the user transitioned from diagramming to a secondary task of chat. Since a notification was pending, and the system detected this moment as a breakpoint, the notification was delivered.

Procedure

Upon arrival at the lab, we went through an informed consent process with the user. The user was provided with a description of the task and allowed to ask any questions. Users were contacted prior to their scheduled session so that the local machine could be configured with their favorite applications and bookmarks as best as possible.

The user was informed that during the task, notifications containing relevant or potentially useful information would occasionally appear. They were asked to select the notification whenever they noticed it and/or the task allowed. If selected, a dialogue box would immediately open, asking the user to rate his or her frustration with having received the notification at that moment. Once the rating was made, a Web page opened showing the full content of the notification. Users were then free to proceed as desired. The task session lasted for 2 hours.

Afterward, a post experiment interview was conducted. The experimenter launched a tool that showed the user's interaction video along with the locations of the system-identified breakpoints. The experimenter navigated to each breakpoint and asked the user to agree or disagree with whether that moment was a breakpoint. If agreed, the user identified its type (coarse, medium, or fine) based on given descriptions. If disagreed, the user scrubbed the video to identify the closest point where they would have preferred to receive the notification and explain why.

Measurements

The following measurements were taken:

- *Frustration*. The rating was made using a 7-point Likert scale, ranging from very pleasing to very frustrating.
- *Reaction time.* This was measured as the time between when a notification appeared and the user selected it.
- *Resumption time.* This was the time from when the user closed the notification content page to when focus on suspended activity was resumed. This time would also include diversions into other activities, if any.

These metrics have been used to measure the effects of interruption in prior work (e.g., [3, 20, 23]). In addition, we solicited feedback on how users felt about having notifications deferred until breakpoints. We also analyzed interaction videos to compare how users responded to notifications delivered under different scheduling policies.

RESULTS

Out of a maximum possible 256 notifications, 64 were to be delivered as Immediate and 192 were to be delivered as Scheduled. Out of the 64 Immediate, 1 was not delivered as the user had already finished the task.

Out of the 192 Scheduled, 170 were delivered. 109 of these were delivered at moments that users agreed were breakpoints, resulting in 64% precision (percentage of predicted moments that were actually breakpoints). Of these 109, users identified 26 as Coarse, 44 as Medium and 39 as Fine. These user-specified types were used as the values for the Policy factor in our analysis. The 61 cases where there was no agreement were excluded, since they are similar to the data collected for Immediate.

The remaining 22 Scheduled could not be delivered as the system could not detect any breakpoint between when the notifications were generated and the end of the task session. These were excluded. Also, in some cases, a notification appeared while the user was still responding to another. This resulted in an additional 29 notifications being excluded to avoid confounding effects. In sum, our rigorous filtering process left 143 data points for analysis.

For Scheduled notifications that were delivered, the mean deferral time (the time from when they were generated to when they were delivered) was 88.6s (S.D. 139.3s). These times are consistent with breakpoint distances reported in [22], and with transition times to non-busy states in [14].

User Reactions and Behavioral Responses

The concept of scheduling notifications at breakpoints was well received by users and matched what they themselves preferred. For example, when scrubbing the video to select preferred moments to receive notifications that had not appeared at a breakpoint, they almost always identified a moment that indicated the completion of an action. This is exemplified in many of their explanations:

> "After I have added this room [would have been a good moment]"
>
> "I wish it had waited until I was done with this [the stairs]"
>
> "Just before that - where I scrolled down the window …just when I ended this method."
>
> "I would have preferred it [the notification] when I had just finished this line."

We also discovered that there was an interaction effect between scheduling policy and notification content. For example, users expressed wanting notifications relevant to their ongoing activity to be delivered at Medium or Fine breakpoints as opposed to Coarse. Even though this may cause higher localized costs (e.g., in terms of resumption lag [21] or reaction time), users perceive a larger global benefit because the notification is received when its content can be best utilized, and precludes the need for a context switch. When a relevant notification was delivered at Coarse, we often observed users immediately returning to the activity they had just left. Users expressed they disliked these occurrences since they were intending to move away from the ongoing task. Receiving a relevant notification caused them to abandon that task switch.

For General Interest notifications, users stated that they wanted them to be delivered only at Coarse. This was also evident in their task behavior. For example, if a general interest notification appeared at Medium or Fine, users cursorily read the content and attempted to return to the suspended task as soon as possible. If it appeared at Coarse, users would often read the content in its entirety, and then proceed with their intended task switch.

The key implication of these results is that notifications deemed relevant to the ongoing activity should be scheduled at Medium or Fine, while notifications of general interest should be scheduled at Coarse. This also indicates that notification scheduling systems must be able to detect all three types of breakpoints in practice.

A related but less commonly observed behavior was users initiating *chains of diversion* [23]. This refers to the activities that a user performs after having attended to a notification but before resuming their suspended task.

25 chains of diversion were observed, 11 for diagram editing and 14 for programming. The nature of the diversion was a function of both the ongoing task and the notification content. For example, during diagram editing, *general interest* notifications caused users to enter a chain of diversion most often (9 of 11). During these diversions, users would check mail, weather or movie schedules, or browse online news. For programming, users went on a chain of diversion most often (11 of 14) after having received a *relevant* notification. Diverted activities were often related to the programming task, e.g., looking up

code samples or browsing online forums. Policy did not seem to have an impact on the chain of diversion.

Frustration

A 3-way ANOVA showed main effects of Content ($F(1, 109)=13.9$, $p<0.001$) and Policy ($F(3, 109)=5.4$, $p<0.002$), and an interaction effect between Policy and Activity ($F(3, 109)=4.7$, $p<0.004$).

For Content, notifications that were of general interest caused more frustration ($\mu=4.98$, S.D.=1.81) than those that were relevant ($\mu=3.59$, S.D.=1.71). Several users stated that even if a notification may have been initially perceived as disruptive, if they determined the content to be relevant, they were more tolerant towards it. This finding is consistent with results in prior work [5, 10].

Due to the interaction, we examined effects of Policy within each Activity separately. For diagram editing, Policy had a main effect on frustration ($F(3, 52)=6.2$, $p<0.001$). Post hoc tests showed that notifications delivered at Coarse ($\mu_C=3.6$, S.D.=1.99) caused lower frustration than at Fine ($\mu_F=5.5$, S.D.=1.7). Notifications delivered at Medium ($\mu_M=2.6$, S.D.=1.6) caused lower frustration than at Fine ($p<0.001$) and Immediate ($\mu_I=4.5$, S.D.=1.58; $p<0.037$). No other differences were found.

For programming, trends were in the expected direction ($\mu_C=3.28$, $\mu_M=4.39$, $\mu_F=4.33$; $\mu_I=4.82$), but did not reach a level of significance. The lack of effect may be due to the fact that programming induced higher cognitive demands than diagram editing, causing users to experience similar levels of frustration across policies. This is further supported by the fact that 15 of the 16 notifications that users failed to respond to were during programming.

Reaction Time

A 3-way ANOVA did not reveal main effects of the factors on reaction time. However, inspection of the graph showed a very salient pattern in how users were reacting to relevant notifications delivered at breakpoints versus immediate (see figure 3). To explore this further, we collapsed breakpoints into a single Breakpoint level and reran the ANOVA for only relevant notifications.

Results showed a main effect of Policy (Breakpoint, Immediate) on reaction time ($F(1, 66)=3.78$, $p<0.056$). Users reacted to notifications at Breakpoints ($\mu=3.07$s, S.D.=1.2) faster than at Immediate ($\mu_I=4.08$s, S.D.=3.13).

A plausible explanation for this outcome is that for Immediate, users needed to first externalize information (e.g., finish the current line of code or complete alignment of shapes) into the task environment, thus causing slower reaction time. When delivered at breakpoints, users could switch their attention more readily to the notification, resulting in faster reaction time. Analysis of user behavior in the videos confirmed the veracity of this explanation.

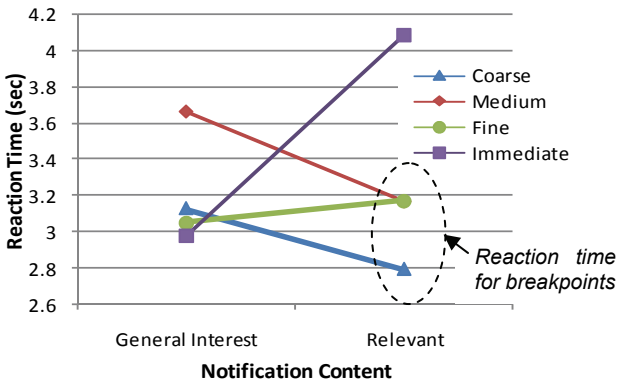

Figure 3. Effects of Policy on reaction time. For Relevant, reaction times were faster for notifications scheduled at breakpoints compared to those delivered immediately.

The same pattern was not found for notifications that were of general interest. This may be because users did not anticipate being away from the task for long, therefore were less concerned about externalizing information.

Resumption Time

An ANOVA revealed an interaction between Content and Activity for resumption time ($F(1,109)=7.75$, $p<0.006$). For Diagram Editing, users resumed suspended tasks faster after responding to notifications that were relevant ($\mu=4.65$s, S.D.=4.4s) compared to those that were of general interest ($\mu=23.1$s, S.D.=41.4s; $F(1,54)=5.91$, $p<0.018$). This result can be attributed to users initiating more chains of diversion after receiving general interest notifications, as previously discussed (see User Reaction).

For programming, users resumed suspended tasks faster after responding to notifications that were of general interest ($\mu=8.8$, S.D.=21.1) compared to those that were relevant ($\mu=16.6$, S.D.=21.5). Differences were due to users having more chains of diversions after receiving relevant notifications, though these did not reach significance. Overall, these quantitative results reflect the qualitative observations discussed under User Reactions.

DISCUSSION

A central goal of this study was to evaluate the impact of using a notification management system to schedule notifications on users and their tasks. Our results showed that users experience meaningfully lower frustration when notifications are scheduled to occur at breakpoints than when delivered immediately. Scheduling notifications at Coarse and Medium results in lower frustration than when scheduled at Fine. An explanation is that users may experience temporary reduction in memory load at these moments, or are at a transition in their action sequence. Our results on frustration are consistent with [20].

Users reacted faster to notifications that were scheduled at breakpoints. For notifications delivered immediately, users would typically externalize their current thought

into the task environment or complete their current action before responding. Similar observations have been noted in [8, 23]. Interestingly, this behavior was not observed when notifications were scheduled at breakpoints. This is likely due to users having just completed their current thought or action at that moment. This difference in user reaction was most apparent for relevant notifications, and seldom surfaced for general interest notifications. One explanation stems from the fact that users would quickly dismiss general interest notifications, regardless of how they were scheduled. Perhaps because users anticipated only being away from the task for a short time, they did not perceive the need to externalize their current thought.

Reductions in frustration and reaction time must be balanced against the time that notifications are deferred. Our results show that the average deferral time was less than 90s. We believe this provides an acceptable balance.

Our results did not show that scheduling notifications at breakpoints affects users' resumption time. Results did show, however, that resumption time depends upon the relevance of a notification to the user's ongoing activity. This is often due to users following *chains of diversion*. For example, for diagram editing, notifications of general interest caused users to initiate chains of diversion most often, whereas for programming, it was the relevant notifications that caused these diversions. One implication is that task reminder tools (e.g., those discussed in [23]) can use the relevance of a notification to help detect whether a user is following a chain of diversion or not.

Another important finding is that users prefer having notifications scheduled at breakpoints. The reason is that this technique closely reflects their own preference for how notifications should be managed. For example, when retrospectively selecting preferred moments for receiving notifications, users identified moments that represented the end of an action corresponding to the completion of a cognitive chunk, e.g., the end of a series of code edits. This strongly indicates that users would accept systems that schedule notifications at breakpoints in practice.

Our results provide further insights into how applications should utilize defer-to-breakpoint policies. For example, applications that generate notifications that are relevant to the user's ongoing activity should request that they be delivered at Medium or Fine breakpoints. This would allow notifications to be delivered when they have the most utility (i.e., *during* the activity), but at less disruptive moments. In contrast, for notifications of general interest, applications should request that they be delivered at Coarse breakpoints. These would be the moments when delivering such notifications would be least disruptive. Urgency of notification content should also be considered when selecting an appropriate policy and timeframe [14].

A second goal of this work was to test the performance of our system in detecting and differentiating breakpoints within novel task sequences. In Study 1, results showed that using composite statistical models in the system can detect breakpoints with 52% (diagram editing) and 59% (programming) recall for novel task sequences. In Study 2, using the models resulted in a precision of 64%. These are promising results since the breakpoints were the ones that users themselves identified in their own tasks. This is an important outcome as it shows that composite models can detect breakpoints for different users performing the same type of complex task with reasonable accuracy.

However, the models performed poorly for differentiating breakpoint *type*. For example, in Study 1, the models differentiated breakpoints with only 2-42% accuracy. Applying the models to differentiate breakpoints in Study 2 did not yield better results. One implication is that systems may want to use such models only for *detecting* breakpoints, i.e., without differentiation. This can be done with modest accuracy, but our results show that users can benefit in terms of reduced frustration and reaction time.

More flexible scheduling policies can be offered if the type of breakpoint could be differentiated. These policies would be useful, e.g., to allow notifications to be more effectively scheduled based on their relevance. Having various policies available would also allow applications to choose an appropriate balance between notification timeliness and costs of interruption for the user. Further work is needed to understand how to improve models for differentiating breakpoints. Related work on modeling interruptibility may provide applicable insights [7, 8, 13].

Two complex task domains were used for this research – diagram editing and programming. Both domains require some form of content generation. Our results are thus most applicable to domains with similar characteristics, e.g., document editing, image manipulation and electronic communication. Future work should study the effects of notification scheduling within other types of task domains such as information-seeking and data manipulation.

CONCLUSION AND FUTURE WORK

Research continues to move closer to enabling intelligent management of notifications for end users. A fundamental challenge is to understand the effects that this type of management would have on users and their tasks.

Our work has made several contributions addressing this challenge. First, we have designed and implemented a novel notification management system that can schedule notifications to occur at any of three types of breakpoints during interactive tasks. The system monitors user interactions and identifies breakpoints through the use of statistical models developed a priori.

Second, we conducted one of the first user studies investigating the effects of notification management. One

promising method was studied - deferring notifications *until breakpoints*. We found that this method yields lower frustration and faster reaction time compared to delivering notifications immediately. This method was also found to be consistent with how users prefer notifications to be managed. New insights were offered for how applications can utilize the relevance of their notifications' content to more effectively select defer-to-breakpoint policies.

Third, we studied how well composite statistical models detect breakpoints within novel task sequences. We found that these models can detect breakpoints reasonably well, but struggle to differentiate their type. Systems that are only able to *detect* breakpoints can benefit users (e.g. reduced frustration and reaction time), but being able to differentiate breakpoint type would offer more flexibility. Our work shows that this flexibility would be useful, e.g., to better balance timeliness and levels of frustration.

For future work, we intend to deploy our system and study its impact over a longer period of time. This should reveal new insights into how notification management systems can be designed to be more effective. Second, we would like to develop models for additional commonly used applications so that more users can realize the benefits of intelligent notification scheduling. Finally, we would like to investigate techniques for improving the ability of models to detect and differentiate breakpoints.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under award no. IIS 05-34462.

REFERENCES

1. Adamczyk, P.D. and Bailey, B.P. If Not Now When? The Effects of Interruptions at Different Moments within Task Execution. *Proc. CHI 2004*, 271-278.
2. Bailey, B.P., Adamczyk, P.D., Chang, T.Y. and Chilson, N.A. 2006. A Framework for Specifying and Monitoring User Tasks. *Journal of Computers in Human Behavior*, *22* (4): 685-708.
3. 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. *Journal of Computers in Human Behavior*, *22* (4): 709-732.
4. Begole, J.B., Matsakis, N.E. and Tang, J.C. Lilsys: Sensing Unavailability. *Proc. CSCW 2004*, 511-514.
5. Czerwinski, M., Cutrell, E. and Horvitz, E. Instant Messaging: Effects of Relevance and Timing. In *People and Computers XIV: Proc. HCI 2000*, 71-76.
6. Fogarty, J. and Hudson, S.E. Toolkit Support for Developing and Deploying Sensor-Based Statistical Models of Human Situations. *Proc. CHI 2007*, 135-144
7. Fogarty, J., Hudson, S.E. and Lai, J. Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. *Proc. CHI 2004*, 207-214.
8. Fogarty, J., Ko, A.J., Aung, H.H., Golden, E., Tang, K.P. and Hudson, S.E. Examining Task Engagement in Sensor-Based Statistical Models of Human Interruptibility. *Proc. CHI 2005*, 331-340.
9. Fogarty, J., Lai, J. and Christensen, J. 2004. Presence Versus Availability: The Design and Evaluation of a Context-Aware Communication Client. *International Journal of Human-Computer Studies*, *61* (3): 299-317.
10. Gluck, J., Bunt, A. and McGrenere, J. Matching Attentional Draw with Utility in Interruption. *Proc. CHI* 2007, 41-50.
11. Ho, J. and Intille, S.S. Using Context-Aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. *Proc. CHI 2005*, 909-918.
12. Horvitz, E. Principles of Mixed-Initiative User Interfaces. *Proc. CHI 1999*, 159-166.
13. Horvitz, E. and Apacible, J. Learning and Reasoning About Interruption. *Proc. ICMI 2003*, 20-27.
14. Horvitz, E., Apacible, J. and Subramani, M. Balancing Awareness and Interruption: Investigation of Notification Deferral Policies. *Proc. User Modeling 2005*, 433-437.
15. Horvitz, E., Breese, J., Heckerman, D., Hovel, D. and Rommelse, K. The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *Proc. UAI 1998*, 256-265.
16. Horvitz, E., Jacobs, A. and Hovel, D. Attention-Sensitive Alerting. *Proc. UAI 1999*, 305-313.
17. Horvitz, E., Kadie, C.M., Paek, T. and Hovel, D. 2003. Models of Attention in Computing and Communications: From Principles to Applications. *CACM*, *46* (3): 52-59.
18. Horvitz, E., Koch, P., Kadie, C.M. and Jacobs, A. Coordinate: Probabilistic Forecasting of Presence and Availability. *Proc. UAI 2002*, 224-233.
19. Hudson, S.E., Fogarty, J., Atkeson, C.G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J.C. and Yang, J. Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study. *Proc. CHI 2003*, 257-264.
20. Iqbal, S.T. and Bailey, B.P. Investigating the Effectiveness of Mental Workload as a Predictor of Opportune Moments for Interruption. *Proc. CHI 2005*, 1489-1492.
21. Iqbal, S.T. and Bailey, B.P. Leveraging Characteristics of Task Structure to Predict Costs of Interruption. *Proc. CHI 2006*, 741-750.
22. Iqbal, S.T. and Bailey, B.P. Understanding and Developing Models for Detecting and Differentiating Breakpoints During Interactive Tasks. *Proc. CHI 2007*, 697-706.
23. Iqbal, S.T. and Horvitz, E. Disruption and Recovery of Computing Tasks: Field Study, Analysis and Directions. *Proc. CHI 2007*, 677-686.
24. McFarlane, D.C. 2002. Comparison of Four Primary Methods for Coordinating the Interruption of People in Human-Computer Interaction. *HCI*, *17* (1): 63-139.
25. Monk, C.A., Boehm-Davis, D.A. and Trafton, J.G. 2004. Recovering from Interruptions: Implications for Driver Distraction Research. *Human Factors*, *46* (4): 650-663.
26. Newtson, D. 1973. Attribution and the Unit of Perception of Ongoing Behavior. *Journal of Personality and Social Psychology*, *28* (1): 28-38.
27. Zacks, J.M. and Tversky, B. 2001. Event Structure in Perception and Conception. *Psychological Bulletin*, *127*, 3-21.