Understanding and Developing Models for Detecting and Differentiating Breakpoints during Interactive Tasks

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

ABSTRACT
The ability to detect and differentiate breakpoints during task execution is critical for enabling defer-to-breakpoint policies within interruption management. In this work, we examine the feasibility of building statistical models that can detect and differentiate three granularities (types) of perceptually meaningful breakpoints during task execution, without having to recognize the underlying tasks. We collected ecological samples of task execution data, and asked observers to review the interaction in the collected videos and identify any perceived breakpoints and their type. Statistical methods were applied to learn models that map features of the interaction to each type of breakpoint. Results showed that the models were able to detect and differentiate breakpoints with reasonably high accuracy across tasks. Among many uses, our resulting models can enable interruption management systems to better realize defer-to-breakpoint policies for interactive, free-form tasks.

CATEGORIES AND SUBJECT DESCRIPTORS
H.1.2 [Models and Principles]: User/Machine Systems – human information processing and human factors

KEYWORDS
Attention, Breakpoints, Interruption, and Statistical models.

INTRODUCTION

A breakpoint is the moment between two meaningful units of task execution [23], and reflects internal transitions in perception or cognition [29]. In the area of interruption management, studies have shown that deferring delivery of notifications until a breakpoint is reached can meaningfully reduce costs of interruption [1, 5, 7, 18, 19]. However, to be able to automate these types of defer-to-breakpoint policies within systems for interruption management, we need to better understand how to efficiently and accurately detect breakpoints during execution of interactive tasks.

CHI 2007, April 28–May 3, 2007, San Jose, California, USA.

One common method for detecting breakpoints is to match users' ongoing interaction to specifications of tasks defined a priori [4]. Although this allows breakpoints to be easily detected within tasks that are fairly prescribed, it is much more difficult to leverage these types of static specifications to detect breakpoints within tasks that have highly variable interaction, i.e., *free-form* tasks, yet these are by far the most common type of computing task performed [8]. This limitation severely inhibits the ability to realize defer-to-breakpoint policies in practice, though these policies have been shown to reduce costs of interruption [1, 5, 7].

In this work, we seek to overcome this central limitation by understanding how to detect breakpoints and differentiate their granularity without requiring any task specification. *Granularity* refers to the degree of perceptual difference of the actions surrounding a breakpoint [28], and the ability to differentiate granularity is critical. For example, this would allow systems to reason about whether to defer notifications until coarser breakpoints, which occur less often, but offer larger reductions in cost; or until finer breakpoints, which occur more often, but offer smaller reductions in cost [5].

A basic question is how many granularities of breakpoints are detectable and meaningful during task execution. From studies of event perception [28, 29] and task interruption [11, 19], there is evidence for at least three perceptually meaningful granularities; *Coarse*, *Medium*, and *Fine*. For example, when editing documents, Fine may be switching paragraphs; Medium may be switching documents; and Coarse may be switching to an activity other than editing.

We investigate how these three granularities of breakpoints are manifested during the execution of free-form tasks and examine the feasibility of building statistical models that can detect and differentiate them. We collected ecological samples of task execution data from three task categories; document editing, image manipulation, and programming. Leveraging methods used to study human perception [23], observers were asked to review collected videos, identify perceived breakpoints and their type, and enter rationale. Breakpoints were thus detected based only on the users' observable interaction, not their internal state, similar to the data that would be available to a system in practice. By aggregating and filtering the breakpoints, we could identify the 'true' breakpoints, i.e., those with high agreement.

From observers' rationale, our own analysis of the data, and related work [11, 19], we identified candidate features of the interaction that might indicate each type of breakpoint. Predictive features were identified from the candidate set and statistical models that map these features to the true breakpoints were learned and evaluated. Results showed that the models were able to detect and differentiate each breakpoint type with reasonably high accuracy across tasks.

The benefit of our models is that they are able to detect and differentiate breakpoints using only features of the ongoing interaction in free-form tasks, without any specifications of those tasks. The use of our models can thus enable systems to better realize defer-to-breakpoint policies in practice.

RELATED WORK

We describe breakpoints, their use in studying perception and action, and the implications for our work; describe how breakpoints can be used; and discuss how our work differs from existing methods of detecting breakpoints.

Breakpoints in Perception and Action

A breakpoint represents the moment of transition between two observable, meaningful units of task execution [23]; and reflects internal transitions in perception or cognition [29]. For example, breakpoints are often used to study how people segment incoming sensory stimuli [14, 23-25, 28, 29]. A method shared in many of these experiments is to have observers review videos of other people performing goal-directed tasks (e.g. repairing a musical instrument) and annotate where they believe one meaningful unit of action ends and the next one begins, i.e., the breakpoints [23]. A consistent finding is that observers identify many of the same locations as breakpoints, showing that perception is segmented into discrete units and a shared cognitive schema is driving this process [28, 29]. These results generally hold for tasks that are familiar and unfamiliar to observers [28].

Observers report that certain visual cues such as changes in the attended-to object, action on that object, or tempo of the action provide salient indicators of breakpoints [29]. This implies that it should be possible to build models (thinking of models as observers) that utilize analogous cues to detect breakpoints within execution of interactive or other tasks.

Another relevant finding is that observers can dynamically modulate the granularity of segmentation between coarse and fine units of action [23], where granularity refers to the degree of perceptual difference between those units. Since a subset of the fine breakpoints typically align with the coarse breakpoints, mental schemas driving perception and action are thought to have at least a two-level hierarchy [28]. This implies that models should differentiate at least two types of breakpoints – Coarse and Fine – during task execution.

Neuroscience studies also show that breakpoints identified by observers of actions are similar to those experienced by the person performing the same actions [26]. This is logical, since the person performing an action is also an observer of their own action as part of a closed loop system [6].

Our work leverages the knowledge and methodology used in this area of research to better understand how to identify perceptually meaningful breakpoints during execution of *interactive* tasks and how to build models that detect them.

How Breakpoints Can Be Used

The ability to detect breakpoints during task execution has many useful applications. For example, for interruption management, studies have shown that deferring delivery of notifications until breakpoints are reached can meaningfully reduce costs of interruption [1, 5, 16, 19], and that deferring until coarser breakpoints further reduces these costs [19]. In these studies, specifications for tasks were determined in advance using modeling techniques such as GOMS [20], enabling interruptions to be cued at specific moments [19]. If breakpoints could be reliably detected in free-form tasks, then defer-to-breakpoint policies similar to those used in the controlled studies could be better realized in practice.

Detection of breakpoints can also contribute to an emerging class of interactive tools that enables knowledge activities to be organized into reusable structures and shared [9, 27]. A challenge in building these types of tools is being able to organize user activities without having to repeatedly solicit input [9]. Models that detect breakpoints could facilitate automated organization, thus reducing the burden on users.

Methods for Detecting Breakpoints

Several methods have been used to detect breakpoints. One common method is to create a structural decomposition of a task, e.g., using GOMS [6], and identify interactions that indicate the end of one subtask and the start of the next – with breakpoints being in-between. As tasks are performed, a system can match users' interaction to the corresponding task descriptions in order to detect the breakpoints [4].

In contrast, our work detects breakpoints using *perceptual structure*. Breakpoints identified using our method would ostensibly be only a subset of those available within the structural decomposition of a task, but *eliminates* the need to create such decompositions and may better identify those breakpoints that typically correspond with lower cost [19].

A second method is to use the number of application-level windows selected within a sliding time window as an indicator of an activity switch [22] (which, as we will show, maps to Coarse breakpoints). Results of this method ranged from 20% to 90% accuracy, but only one type of breakpoint could be detected. A third method is to detect switches between 'rooms' in a virtual desktop window manager [15]. This method could only detect a single type of breakpoint and would force the use of this type of window manager.

A related thread of research has produced statistical models of interruptibility for interactive tasks [11]. One explanation

as to why these models work is that they implicitly detect a user's time to breakpoint. Indeed, the authors reported that users would often defer acceptance of a cued notification until they reached a breakpoint. Our work explicitly detects these points of interest, which could be used for interruption management as well as other purposes (see prior section).

In the area of mobile devices, Ho and Intille [16] used data from multiple accelerometers attached to a person's body. Based on analyzing the data signatures in various physical postures and movements, they were able to detect moments when users were in physical transition (such as the act of standing up). Results from a study showed that deferring an interruption until this transition reduced cost of interruption.

Relative to this corpus of research, our work is original in that we focus on building models that are able to detect and differentiate *three* granularities of perceptually meaningful breakpoints within interactive, free-form tasks; and without any specifications. The methods used in our work could also be applied to build effective models for detecting and differentiating breakpoints within physical or other tasks.

OVERVIEW OF THE MODEL BUILDING PROCESS

To develop effective and efficient models for detecting and differentiating task breakpoints, our process was to:

- Collect representative samples of users' task execution, in the form of screen interaction videos and event logs.
- Have observers review the videos, identify perceived breakpoints and their type, and explain their rationale.
- Select those breakpoints with a high degree of agreement, and use them as the ground truth for building the models.
- Identify features describing the interaction at the selected breakpoints, guided by users' explanations, and compute values for the features based on the videos and logs.
- Learn statistical models that map the predictive features to the ground truth values, and evaluate their accuracy.

To facilitate collection and analysis of the breakpoint data, we developed several new software tools. Activity Recorder records a user's screen interaction and logs system events; Breakpoint Annotator enables observers to review videos, identify breakpoints, and enter linguistic explanations; and Breakpoint Analyzer supports interactive analysis of the data. Our tools can be utilized to reduce the effort required to collect and analyze similar data, e.g., data in [23, 25, 28].

COLLECT TASK EXECUTION DATA

Task execution data was collected from three general task categories; Document Editing (DE), Image Manipulation (IM) and Programming (P). These categories were selected because they are often performed by many users, comprise diverse subtasks, and require varying engagement. Using several categories would allow better understanding of the similarities and differences among breakpoints across tasks.

For each category, two users (6 total) were recruited and screened to ensure they were experienced in the category selected and would be comfortable having their interaction data viewed by others. Users received $20 for participating.

We wanted to collect samples of users' own personal or work tasks, performed in their own environment, ensuring a high degree of ecological validity. Our recording software was thus installed on users' own machines and they were informed of what data it was recording and how to control it. For example, the software allows recording to be started, paused, or stopped at any time using keyboard shortcuts and shows its current status through an icon in the system tray.

The software was configured to record screen interaction at a low, but adequate frame rate (5 fps) using the Camtasia SDK and logged mouse, keyboard, and other relevant system events using the Windows Hooks API. Users were asked to activate the recording software the next time that they would be primarily focused on performing any task within the relevant category for at least an hour. We emphasized that they should perform the task, with the interleaving of any other tasks, as usual. To avoid recording sensitive data, users were reminded that they could pause/ restart the software at any time. Once at least an hour of data was recorded (minus any pauses), the user notified the experimenter, who collected it and removed the software.

For task content, for DE, one user was writing a research paper while the other was writing study guides for exams. For IM, one user was touching up personal photos from a recent vacation while the other was developing icons and other graphics for a software application. For P, one user was developing a user interface for a research project while the other was writing source code for a course assignment. The applications used included Microsoft Word, Adobe Photoshop, and Eclipse, respectively. Users did temporarily pause collection of their data, but this was very rare overall.

IDENTIFY PERCEIVED BREAKPOINTS AND THEIR TYPE

The next step was to determine the locations of perceived breakpoints and their type within the task execution data. 24 observers were recruited, 8 per category, and were asked to review the two videos from an assigned category, mark the location and type of each perceived breakpoint, and enter a brief description as to why they felt this was a breakpoint.

Observers were asked to detect and differentiate three types of breakpoints, guided by the following descriptions:

- *Coarse*. The moment when the largest meaningful and natural unit of execution ends and the next one begins.
- *Fine*. The moment when the smallest meaningful and natural unit of execution ends and the next one begins.
- *Medium*. The moment when a natural and meaningful unit of execution, which is smaller than Coarse but larger than Fine, ends and the next one begins.

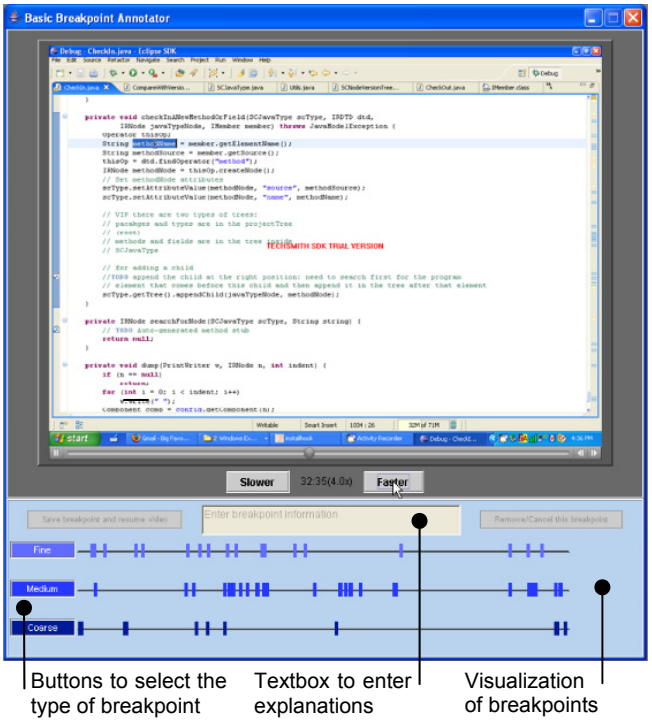

Figure 1. Screenshot of the Breakpoint Annotator tool being used to annotate one of the Programming task execution videos.

Inclusion of Coarse and Fine breakpoints, along with their descriptions, is consistent with research on event perception [23, 28]. Medium was included since empirical studies have shown three classes of interruption cost [11, 19], ostensibly tied to three levels of breakpoints, and results from a pilot study showed that users were able to differentiate the three types of breakpoints within data samples, but not more.

Using observers to identify breakpoints in *another* user's tasks is effective because research has shown that the same schema used to chunk a person's goal-directed actions are also used to chunk their perception when observing another person performing those same actions [26]. Also, finding that observers are able to agree on the types and locations of breakpoints would indicate that similar salient cues were being perceived within the interaction data. If those cues could be identified, then models could be built (thinking of models as observers) that automate a similar process.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	184	226	132
	DE2	140	209	212
Image manipulation	IM1	93	120	293
	IM2	37	99	282
Programming	P1	50	176	193
	P2	252	220	156
Total		756	1050	1268

Table 1. Frequency distribution of breakpoints across tasks.

Category	Breakpoint	Next Coarse	Next Medium	Next Fine
Document Editing	Coarse	141 (235)	192 (283)	259 (289)
	Medium	191 (233)	102 (190)	253 (426)
	Fine	259 (367)	175 (356)	112 (231)
Image Manipulation	Coarse	266 (520)	300 (254)	113 (106)
	Medium	538 (564)	244 (330)	117 (167)
	Fine	641 (663)	380 (421)	91 (115)
Programming	Coarse	162 (397)	116 (151)	174 (168)
	Medium	427 (670)	129 (173)	157 (159)
	Fine	402 (623)	142 (157)	139 (186)
Overall Averages	Coarse	190 (365)	203 (239)	182 (219)
	Medium	385 (512)	158 (226)	176 (306)
	Fine	434 (591)	179 (397)	114 (174)

Table 2. Mean distances in seconds between adjacent types of breakpoints. Standard deviations are in parenthesis.

For procedure, observers came to our lab and were asked to review videos of task execution and identify moments at which they felt that one unit of execution ended and another began; using cursor movements, interaction sequences, and state of the task as cues. The different types of breakpoints were explained using the previous descriptions. The overall methodology was consistent with prior work [23, 24, 29].

Our Breakpoint Annotator tool (Figure 1) was used to assist the observer in the annotation process. The observer was given a demonstration of the tool and practiced using it on a sample of the data, enabling her to become familiar with the interface and 3 types of breakpoints. Once questions were answered, the observer began annotating the first video.

When a breakpoint was detected, the observer selected a button indicating the type of breakpoint (Coarse, Medium, or Fine). In response, the video was paused, a tick mark was shown on the relevant timeline, and a textbox was activated for entering an explanation. The observer could review the video and modify breakpoints as desired. The observer annotated both videos within an assigned category, but since annotation required about two hours, the process was split across two days. The order of videos in a category was counter-balanced. Observers received $20 for participating.

Summary and Characteristics of Breakpoints

A total of 3074 breakpoints (Coarse=756, Medium=1050, Fine=1268) were identified, and are summarized in Table 1. Overall, Fine breakpoints were the most frequent while Coarse breakpoints were the least frequent ($\chi^2(2)=128.9$, $p<0.001$); showing that interactive tasks also tend to be performed in a hierarchical manner [28]. Interestingly, the distributions for tasks DE1 and P2 show more Coarse and Medium breakpoints than Fine breakpoints. This is not unexpected given users' constant multi-tasking behavior [8, 12], which, as our results show, may not always be uniform.

Temporal distances between breakpoints are summarized in Table 2. The average distance between breakpoints ranges from about 1.5 min (between Fine breakpoints for IM) to 10.7 min (from Fine to Coarse for IM), with the overall average between any two breakpoints being about 3.8 min. These results support and extend data reported in [12, 21].

This data is important because it provides some of the first ecological estimates of how long an interruption reasoning system would need to defer delivery of information in order to reduce interruption cost. For example, assuming that information became available just after a user crossed a Fine breakpoint, delivery of the information would need to be deferred up to about 2 min to have some reduction in cost (next Fine breakpoint), about 4 min to further reduce cost (next Medium), and about 7 min to have minimal cost (next Coarse). These values could also inform the design of interfaces that allow users to specify how long they would be willing to wait for different types of information [17].

From observers' explanations, Coarse breakpoints typically corresponded to a switch in high-level activity, indicated by switching to other application(s) judged to be unrelated to the main task, e.g., changing to a music player, checking e-mail, or reading news online. A Coarse breakpoint was also often indicated by returning back to the main application.

Medium breakpoints were tied to switching to applications judged to be relevant to the primary task or to a large shift in focus within the content of the application. For example, for DE, this included transitioning to edit a paragraph in another section of the document, saving the document, and opening another document. For IM, this included loading another image, transitioning to edit a different region or visual feature of the image, and saving the current image. For P, this included starting to edit a new class in the file, saving the current source file, switching to another source file, and switching between the code and debug windows.

Fine breakpoints were usually tied to actions on the content within an application. For example, for DE, this included completing formatting commands, searches, and copy/paste sequences; and starting to edit another paragraph near the current insertion point. For IM, this included completing layer manipulations, resize of canvas, and operations such as color adjustments, blending, cropping, and selection. For Programming, this included starting a new method, closing a method, completing a compile, completing the check in/out of a file; and completing definition of class variables.

Interestingly, observers did not identify lower-level units, such as completing a specific sentence or line of code, or moving between fields in a dialog, as Fine breakpoints. The commonly cited reason, clearly evident in the videos, was that editing at the level of a sentence, line of code, or area of pixels exhibited rapid interleaving of pointing, typing, erasing, selecting, scrolling, etc.; thus offering few visually identifiable breaks in the interaction. Thus, attempting to detect breakpoints at this level of detail is probably not warranted, consistent with earlier empirical findings [18].

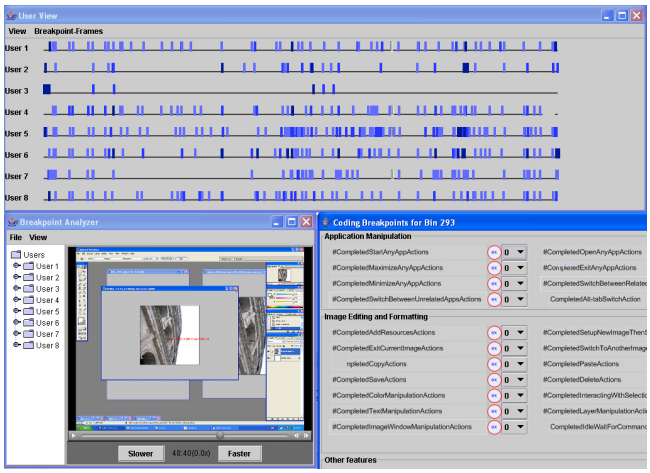

Figure 2. A screenshot of our tool that allows breakpoints to be aggregated (top window) and interactively analyzed. When a breakpoint is selected, the video (bottom left) is positioned at the corresponding temporal location. Candidate features are shown at the right and allow each bin within the video to be quickly coded.

Overall, this data offers some of the first evidence as to where and how often breakpoints occur within interactive tasks, and offers insight into the types of features that might be useful in models for detecting and differentiating them.

IDENTIFY GROUND TRUTH FOR BREAKPOINTS

The third step was to combine the breakpoint data across observers and identify breakpoints that had high agreement. This would remove "noise" from the data set and provide the ground truth for the model building process. Figure 2 shows a screenshot of our interactive tool that was used to facilitate analysis and coding of the breakpoint data.

We first needed to divide the interaction data into discrete bins, which is necessary since there is natural variance in the temporal locations that refer to the same breakpoint, e.g., some observers may take different amounts of time to decide whether a breakpoint had just occurred.

Our goal was to select a bin size large enough such that slightly different locations referring to the same breakpoint would fall into the same bin, but small enough such that locations referring to different breakpoints would not. Whether a marked location referred to the same breakpoint was determined by analyzing observers' explanations and the corresponding parts of the interaction videos and logs.

From testing a number of bin sizes, between 1s and 20s, we found that a bin size of 10s best met our goal and that this value achieved our goal for each type of breakpoint. This is slightly larger than bin sizes used in prior work [14, 24, 28], but our tasks were of much longer duration, on the order of hours as opposed to minutes. Table 3 shows the number of bins for each task, and how many of those bins contained

Category	Task	#Bins	Bins w/ Coarse	Bins w/ Medium	Bins w/ Fine
Document Editing	DE1	360	78	106	85
	DE2	425	60	123	157
Image manipulation	IM1	310	43	85	185
	IM2	434	25	73	160
Programming	P1	406	19	85	126
	P2	371	108	130	110
Total		2306	333	602	823

Table 3. Frequency distribution of bins and number of bins with each type of breakpoint. Each bin represents 10s of task execution.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	4 (2.3,1.8)	3 (2.1,1.2)	2 (1.5,0.8)
	DE2	4 (2.3,1.3)	3 (1.7,1.0)	2 (1.3,0.7)
Image manipulation	IM1	4 (2.2,1.3)	2 (1.4,0.7)	2 (1.6,0.8)
	IM2	2 (1.5,0.7)	2 (1.3,0.7)	3 (1.7,1.3)
Programming	P1	5 (2.6,1.9)	4 (2.1,1.5)	2 (1.5,0.8)
	P2	4 (2.3,1.8)	3 (1.7,1.0)	2 (1.4,0.8)

Table 4. Min number of breakpoints (mean, 1.65*s.d.) that had to be marked within a bin before it was considered a *true* breakpoint.

Category	Task	Coarse	Medium	Fine
Document Editing	DE1	16 (21%)	40 (38%)	35 (41%)
	DE2	11 (18%)	25 (20%)	38 (24%)
Image manipulation	IM1	8 (19%)	24 (28%)	74 (40%)
	IM2	9 (36%)	19 (26%)	29 (18%)
Programming	P1	40 (21%)	10 (12%)	47 (37%)
	P2	22 (20%)	23 (18%)	11 (10%)

Table 5. Distribution of true breakpoints. Percentages indicate what percent of bins (Table 3) satisfied the threshold (Table 4).

each type of breakpoint. If a bin had multiple types of breakpoints, it was counted multiple times. A chi-square test showed that observers were biased towards selecting certain bins as breakpoints across all six tasks (DE-1: $\chi^2(359)=1323$, $p<0.0001$; DE-2: $\chi^2(424)=1208$, $p<0.0001$; IM1: $\chi^2(309)=408$, $p<0.0001$; IM2: $\chi^2(433)=997$, $p<0.0001$; P-1: $\chi^2(405)=1183$, $p<0.0001$; P-2: $\chi^2(370)=957$, $p<0.0001$), meaning that the selection of breakpoints was *not* random.

We then had to establish the minimum number of observers who needed to have indicated that a breakpoint was within a bin before being able to conclude that that bin contained a "true" breakpoint. One solution would be to use an absolute threshold (e.g., more than half of the observers must agree), but this does not consider the prior likelihood of agreement.

Our approach, following [14], was to compute the average number of breakpoints per bin, considering only those bins with at least one breakpoint; add 1.65 standard deviations; and round. This process establishes an alpha=.05 threshold [14], and this threshold was calculated for each task and breakpoint type. A bin with a number of breakpoints (same type) greater than the computed threshold was considered to contain a true breakpoint, or *breakpoint bin*. Table 4 shows the decision thresholds used in this filtering process.

The number of breakpoints meeting the thresholds was 445 (~25% of all bins with $\geq$ 1 breakpoint), and are summarized in Table 5. Inspection of the table shows that the filtering was fairly uniform. Though this was a stringent filtering process, the aim was to reduce the number of false positives in the data set that would later be used for training. Also, independent sample t-tests confirm that more observers had detected a breakpoint in a breakpoint bin than in the other bins across tasks and breakpoint type ($p<0.001$ in all cases).

What is perhaps most intriguing about this result is that the observers, all of whom had annotated the videos separately, identified many of the same moments as breakpoints. This occurred because observers were likely perceiving similar cues in the interaction videos. This implies that it should be possible to build models that leverage those same cues to detect and differentiate breakpoints for free-form tasks.

Though there were fewer breakpoint bins due to filtering, the average temporal distances were similar to those listed in Table 2 and ranged from 1.4 min to 11.9 min, with the average between any two breakpoint bins being 4.3 min.

IDENTIFY FEATURES INDICATING BREAKPOINTS

Next, we needed to identify features that could be used to detect and differentiate breakpoints during task execution. Candidate features were determined based on an analysis of observers' explanations and event logs, our own analysis of the task data, lessons reported in prior work [10, 11], and whether values could be realistically computed in a system.

For Coarse breakpoints, observers were very consistent in describing them as a switch to another activity that was not related to the main task (and back). However, this abstract description does not yield any specific, usable features and a model would not be able to know what a user's main task was without prior knowledge. Based on detailed inspection of video segments corresponding to Coarse breakpoints, we observed that they were frequently tied to switches among various types of applications or content, e.g., music players, e-mail and instant messaging, or online shopping and news. Our observations are also consistent with results derived from an analysis of users' activity data, as reported in [8].

We thus created a set of application categories including Entertainment, Communications, and Web; with the latter being further categorized based on whether it is a common news or shopping site based on its URL; and those already being used as part of this work (DE, IM, and P). Under the assumption that various applications could be mapped to

General (Coarse)	Application Specific (Medium and Fine)		
App Manipulation (20 total)	Document Editing (33 total)	Image Manipulation (33 total)	Programming (42 total)
#CompletedOpenAnyApp **#SwitchToEntertainmentApp** **#SwitchToOnlineNews** **#SwitchToDocEditing** **#SwitchToImageManipulation** **#SwtichToProgramming** #SwitchesToCommunications #CompletedStartAnyApp #CompletedMaximizeAnyApp #CompletedExitAnyApp #CompletedRelocation	**#CompletedFormattingActions** **#CompletedSwitchToDocEditing** **#CompletedAlt-tabSwitch** **someKeystrokes** **noMouseClicks** **noMouseMoves** **someMouseMoves** #CompletedSwitchToAnotherDoc #CompletedSetInsertionPoint #CompletedSelections #CompletedSaves	**#CompletedAltTabSwitch** **#CompletedSwitchToAnotherImg** **#CompletedSave** **#CompletedColorManipulation** **#CompletedTextManipulation** #CompletedSetupNewImage #CompletedExitCurrentImage #CompletedSelectionTools #CompletedLayerManipulation #CompletedCanvasResize #CompletedSelectionToolActions	**#CompletedOpenAnyApp** **#CompletedSearch** **#CompletedSwitchClass** **#CompletedSwitchProject** **#ControlKeyStrokes** **noMouseClicks** #CompletedSetInsertionPoint #CompletedSwitchMethod #CompletedCreateMethod #CompletedCreateClass #CompletedDebug #CompleteNavigateCode

Table 6. A representative sample of the candidate features used for detecting breakpoints. The number of occurrences of each feature were counted for each 10s bin of task execution. The features highlighted in bold were found to be predictive.

these categories, features were created for the number of switches between them. Also, the number of applications started, exited, and moved were included, as these have also been argued to indicate switches in high-level activity [22].

Though our approach offers a reasonable starting point and extends prior work for detecting Coarse breakpoints, future work should explore the value of including features tied to the degree of similarity among application content, e.g., using techniques in [3]. Note that overcoming challenges of applying such techniques within the domain of interactive applications is well beyond the scope of our current work.

Medium and Fine breakpoints typically occurred during the interaction within an application. Our approach here was to bind features to independent actions at the application interface level, following work in [11]. For example, for DE, features included CompletedSwitchToAnotherDoc, CompletedSetInsertionPoint, and CompletedScroll. If the first two occurred within a bin, then this would likely indicate Medium; whereas if the latter two occurred, then this might indicate Fine, e.g., due to switching paragraphs.

For Coarse breakpoints, we identified 20 features that were independent of any one application. For Medium and Fine, we identified 33 features for DE, 33 for IM, and 42 for Programming, with some overlap. Samples of the features (with mnemonic descriptions) are provided in Table 6. One characteristic of many of the features is they correspond to *completion* of an action, not the action itself (e.g. completed scrolling as opposed to scrolling), which is consistent with observers' explanations and the notion of a breakpoint.

A coding agenda was developed, comprising a description, example, and rule for each feature [2]. For each breakpoint bin (10s clip), values for the features were computed by applying the agenda to corresponding parts of the videos. We also computed values for the features for a sample of bins that had *no* breakpoint (NAB), enough to compose 25% of the total training cases. Training cases were in the form of <value of feature 1, …, value of feature N, output>, where output was one of Coarse, Medium, Fine, or NAB.

The coding was validated by having an independent coder compute values for the candidate features for 10% of the bins, randomly selected from the training cases. Cohen's Kappa showed satisfactory agreement between them (0.74).

EXTRACT PREDICTIVE FEATURES

Before predictive features could be extracted, we needed to decide how the models would be built. Our approach was to create one application-independent model for predicting Coarse/NAB and a set of application-specific models for predicting Medium/Fine/NAB, giving a total of 4 models. This decision was made because Coarse breakpoints were deemed independent of any one application while Medium and Fine were more dependent. Training cases were organized accordingly, but Medium and Fine cases from each task category were included as part of NAB cases for Coarse, helping to minimize overlap between the models.

Given this organization of the training cases (models), the predictive features were extracted using Correlated Feature Selection (CFS) with a Greedy Stepwise search [13]. CFS was chosen since some candidate features may have been correlated. Predictive features are shown in bold in Table 6.

MAP PREDICTIVE FEATURES TO BREAKPOINTS

The last step was to learn models that map the predictive features to the breakpoint types and NAB. A multilayer perceptron (MLP) was leveraged to learn each mapping, as it does not assume independence of features and has been used to learn similar models in prior work [19]. The model for Coarse breakpoints had two outputs (Coarse, NAB) while the models for each category of task had three outputs (Medium, Fine, NAB). All models had one hidden layer.

For input, the model for Coarse used only those features that were independent of the task (left column of Table 6) while inputs for the other models corresponded to features tied to the application, in addition to the general features. Mappings were learned using back propagation, and a 10-fold cross validation was used to evaluate the models.

		Predicted		
		Coarse	NAB	Total
Actual	Coarse	62 (89.9%)	7 (10.1%)	69 (100%)
	NAB	11 (15.7%)	59 (84.2%)	70 (100%)

Table 7. Predicted vs. Actual for Coarse breakpoints. Overall accuracy was 87.1%.

Table 7 shows results for predicting Coarse and NAB. The model yielded an overall accuracy of 87.1%, which is much better than the baseline ($\chi^2(1, 139)=76.3$, $p<0.001$; baseline =50%), where baselines were calculated as the accuracy of always predicting the most common outcome. The high accuracy can likely be attributed to the model's features detecting a switch between certain application categories that often indicated a switch between unrelated activities. More sophisticated analysis of the similarity between the content of applications may yield further improvements.

Tables 8a-c show results for detecting and differentiating Medium, Fine, and NAB for the three task categories. For Document Editing, the model yielded an overall accuracy of 69.4%, which is much better than the baseline ($\chi^2(1, 85)=33.5$, $p<0.001$; baseline=39%). The model was slightly less accurate for differentiating between Medium and Fine. However, the most egregious type of error, detecting either type of breakpoint when none existed, was low (14.4%).

For Image Manipulation, the model yielded an accuracy of 76.3%, much better than the baseline ($\chi^2(1, 152)=42.1$, $p<0.001$; base=50%). This model was able to effectively differentiate Medium and Fine, and Medium and NAB. However, the model would sometimes predict Fine when the actual was NAB. This could be due to the mouse movements being less predictive of users' intents or there being less visible structure in this particular task category.

For Programming, the model yielded an accuracy of 75.8%, which was better than the baseline ($\chi^2(1, 91)=23.3$, $p<0.001$; base=51%). The model was slightly less effective at differentiating Fine and NAB, but it was very effective at differentiating Medium and NAB, and Medium and Fine.

Our models were developed using breakpoints identified by observers who did not share users' internal understanding of their tasks. As a final evaluation metric, we thus wanted to test how well our models could predict breakpoints identified by the users themselves. We asked users whose interaction data was originally annotated by observers to identify breakpoints in their own data, and then tested the accuracy of our models on it. Applying our models to the user's annotated data sets, the accuracy of the model for Coarse breakpoints ranged from 40–100%, with an average of 76.5% across users (one user's data was excluded as too few breakpoints were identified). For the application specific models, the results for each user were (DE1: 56.0%, DE2: 72.7%; IM1: 68.2%, IM2: 85.7%; P1: 14%, P2: 50.0%). Other than for P1, these results show that our models were able to accurately predict breakpoints identified by the users, even though a number of these breakpoints did not intersect with those identified by the observers. This validates that our models can predict breakpoints independent of the knowledge of the task.

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	20 (60.6%)	11 (33.3%)	2 (6.1%)	33 (100%)
	Fine	5 (20.8%)	15 (62.5%)	4 (16.6%)	24 (100%)
	NAB	2 (7.2%)	2 (7.2%)	24 (85.7%)	28 (100%)

Table 8a. Predicted vs. actual breakpoints for Document Editing. Overall accuracy was 69.4%

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	24 (68.6%)	10 (28.6%)	1 (2.8%)	35 (100%)
	Fine	5 (6.6%)	71 (93.4%)	0 (0%)	76 (100%)
	NAB	2 (4.9%)	18 (43.9%)	21 (51.2%)	41 (100%)

Table 8b. Predicted vs. actual breakpoints for Image manipulation. Overall accuracy was 76.3%.

		Predicted			
		Medium	Fine	NAB	Total
Actual	Medium	11 (68.8%)	4 (25.0%)	1 (6.3%)	16 (100%)
	Fine	1 (2.2%)	36 (78.2%)	9 (19.6%)	46 (100%)
	NAB	0 (0%)	7 (24.1%)	22 (75.9%)	29 (100%)

Table 8c. Predicted vs. actual breakpoints for Programming. Overall accuracy was 75.8%

Overall, even though there were some errors, our results demonstrate that it is feasible to build models that detect and differentiate breakpoints within free-form tasks with fairly high accuracy. This ability to detect a majority of the breakpoints should be more than sufficient to allow useful functionality, e.g., to enable defer-to-breakpoint policies. Potential solutions for meaningfully improving the accuracy of the models involve identifying and integrating additional predictive features into the models, training the models for specific users, and experimenting with various bin sizes.

DISCUSSION

This research sought to further understand different types of breakpoints across various tasks and examine the feasibility of building models that could detect and differentiate them.

Our work has produced several important findings. First, we were able to identify interactions that characterize each type of breakpoint. For example, a switch in high-level activity corresponds to a Coarse breakpoint, a switch in the

current source object (e.g., document, image, or code file) of an application corresponds to Medium, and a switch in the action on the current object corresponds to Fine. This shows that there is a perceivable structure within free-form tasks, which models should be able to detect. Interestingly, these characteristics closely parallel those found to indicate breakpoints within physical tasks [29].

Second, we found that temporal distances between types of breakpoints ranged from about 1 to 10 min, with an average of about 4 min. Our results support previous work showing that users repeatedly multi-task [12], but also show that this multi-tasking occurs at multiple levels of detail. Our results also establish that breakpoints occur often enough such that interruption management systems could practically employ defer-to-breakpoint policies for non-critical notifications.

Third, we reported which features of user interaction were found to be predictive of each breakpoint type. Though our set of features should by no means be considered final, they do provide deeper insights into the range of features that should be included in similar models deployed in practice.

Finally, our models were able to accurately detect 69 to 87% of each type of breakpoint for the observers' data, and similar results were obtained for users' own annotations. We believe these results are very positive, especially since no pre-defined specifications of tasks were used. Increasing accuracy would likely require identifying and integrating additional predictive features, e.g., indicating similarity of content, or refining the default models for specific users.

Deploying Models for Detecting Breakpoints

To deploy similar models in practice, one must consider (at least) how to instrument applications, which breakpoints to detect and how to train the models, and what bin size to use.

Instrumentation of applications is needed to send relevant events to a model. Such instrumentation can be achieved by leveraging existing research efforts such as [9], intercepting application events by writing plug-ins [5], or adapting the underlying UI toolkit [11]. Regardless of the method used, our work provides valuable insights as to the type and level of detail of the instrumentation needed.

Our work shows that three types of breakpoints can be detected, but this does not mean that models must detect all three in practice. For example, in interruption management, if users are able and willing to have information deferred up to 4-5 min on average, then a system may only need to utilize the one model for detecting Coarse breakpoints.

Default models could be deployed and provide reasonable accuracy, but, if needed, accuracy could be improved by refining models on a per user basis [10]. This could be achieved by leveraging toolkits for generating models on-the-fly [11], assuming users would be willing to provide the necessary input. Further improvements would require identifying and integrating additional predictive features.

Finally, to detect breakpoints, a model must typically assess interaction within a fixed time window. Our work suggests a window of about 10s, but an implementation may need to experiment with different values, considering the tradeoff between computational overhead and discriminatory power.

Limitations

There are several limitations to our work. First, our work investigated breakpoints within categories of tasks that all required the generation or manipulation of content. Future work should thus study models for detecting breakpoints in other tasks, e.g., those that stress information-seeking.

Second, we analyzed about one hour of task execution data from each of six users. Thus, our resulting models are only able to accurately detect breakpoints within the range of interaction that was captured in our original data. As our work has now shown that building statistical models for detecting breakpoints is feasible, more robust models could be built by applying the methodology in this work on a much larger sample of interaction data.

Finally, the cognitive duration of a breakpoint is not known, but would be important for certain applications of our work, e.g., for interruption management where cost may not be reduced if information delivery exceeds this duration after a breakpoint. Future work should try to empirically determine this duration, which may inform the bin size for the models.

CONCLUSION AND FUTURE WORK

The ability to detect and differentiate breakpoints represents an emerging need within at least interruption management, e.g., to enable defer-to-breakpoint policies. Our work has made several contributions addressing this need.

First, we leveraged work in psychology to better understand the concept of breakpoints and leveraged unit identification methodology to identify three granularities of perceptually meaningful breakpoints during task execution. Second, we provided insights into the characteristics of interaction that indicate each breakpoint type and evidence as to how often each type of breakpoint naturally occurs in practice.

Finally, our models were able to detect 69% to 87% of each type of breakpoint across tasks for the observers' data, and similar results were obtained for users' own annotations. Our work has thus demonstrated the feasibility of building statistical models that are able to detect and differentiate perceptually meaningful breakpoints during free-form tasks, without having to create any specifications of those tasks.

Beyond addressing the limitations previously discussed, our main direction of future work is to build similar models as part of a broader system for interruption management. The models would enable various policies to be programmed for deferring delivery of non-critical notifications, e.g., defer until next Coarse breakpoint or until the next breakpoint of any type, and tested with various categories of tasks.

ACKNOWLEDGMENTS

We thank the participants who volunteered time to be in our studies. This work was supported in part by the National Science Foundation under award no. IIS 05-34462.

REFERENCES

1. Adamczyk, P.D. and B.P. Bailey. If Not Now When? The Effects of Interruptions at Different Moments within Task Execution. *CHI*, 2004, 271-278.
2. Altheide, D.L. *Qualitative Media Analysis*. Sage, Newbury Park, CA, 1996.
3. Baeza-Yates, R.A. and B. Ribeiro-Neto *Modern Information Retrieval*. Addison-Wesley, Boston, 1999.
4. Bailey, B.P., P.D. Adamczyk, T.Y. Chang and N.A. Chilson. A Framework for Specifying and Monitoring User Tasks. *Journal of Computers in Human Behavior*, *22* (4), 2006, 685-708.
5. Bailey, B.P. and J.A. Konstan. 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), 2006, 709-732.
6. Card, S., T. Moran and A. Newell *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, 1983.
7. Czerwinski, M., E. Cutrell and E. Horvitz. Instant Messaging: Effects of Relevance and Timing. *People and Computers XIV: Proceedings of HCI*, 2000, 71-76.
8. Czerwinski, M., E. Horvitz and S. Wilhite. A Diary Study of Task Switching and Interruptions. *CHI*, 2004, 175-182.
9. Dragunov, A.N., T.G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li and J.L. Herlocker. Tasktracer: A Desktop Environment to Support Multi-Tasking Knowledge Workers. *Proc. IUI*, 2005, 75-82.
10. Fogarty, J., S.E. Hudson and J. Lai. Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. *CHI*, 2004, 207-214.
11. Fogarty, J., A.J. Ko, H.H. Aung, E. Golden, K.P. Tang and S.E. Hudson. Examining Task Engagement in Sensor-Based Statistical Models of Human Interruptibility. *CHI*, 2005, 331-340.
12. Gonzalez, V.M. and G. Mark. "Constant, Constant, Multi-Tasking Craziness": Managing Multiple Working Spheres. *CHI*, 2004, 113-120.
13. Hall, M.A. Correlation-Based Feature Selection for Discrete and Numeric Class Machine Learning. *Proceedings of the 17th International Conference on Machine Learning*, 2000, 359-366.
14. Hanson, C. and W. Hirst. On the Representation of Events: A Study of Orientation, Recall, and Recognition. *Journal of Experimental Psychology: General*, *118* (2), 1989, 136-147.
15. Henderson, A. and S.K. Card. Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface. *ACM TOG*, *5* (3), 1986, 211-243.
16. Ho, J. and S. Intille. Using Context-Aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. *CHI*, 2005, 909-918.
17. Horvitz, E., P. Koch and J. Apacible. Busybody: Creating and Fielding Personalized Models of the Cost of Interruption. *CSCW*, 2004, 507-510.
18. Iqbal, S.T. and B.P. Bailey. Investigating the Effectiveness of Mental Workload as a Predictor of Opportune Moments for Interruption. *CHI*, 2005, 1489-1492.
19. Iqbal, S.T. and B.P. Bailey. Leveraging Characteristics of Task Structure to Predict Costs of Interruption. *CHI*, 2006, 741-750.
20. John, B.E. and D.E. Kieras. The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM TOCHI*, *3* (4), 1996, 320-351.
21. Mark, G., V.M. Gonzalez and J. Harris. No Task Left Behind? Examining the Nature of Fragmented Work. *CHI*, 2005, 321-330.
22. Nair, R., S. Voida and E. Mynatt. Frequency-Based Detection of Task Switches. *Proceedings of the 19th British HCI Group Annual Conference*, 2005, 94-99.
23. Newtson, D. Attribution and the Unit of Perception of Ongoing Behavior. *Journal of Personality and Social Psychology*, *28* (1), 1973, 28-38.
24. Newtson, D. and G. Engquist. The Perceptual Organization of Ongoing Behavior. *Journal of Experimental Social Psychology*, *12*, 1976, 436-450.
25. Newtson, D., G. Enquist and J. Bois. The Objective Basis of Behavior Units. *Journal of Personality and Social Psychology*, *35* (12), 1977, 847-862.
26. Rizzolatti, G., L. Fadiga, V. Gallese and L. Fogassi. Premotor Cortex and the Recognition of Motor Actions. *Cognitive Brain Research*, *3*, 1996, 131-141.
27. Shen, J., L. Li, T. Dietterich and J. Herlocker. A Hybrid Learning System for Recognizing User Tasks from Desktop Activities and Email Messages. *Proc. IUI*, 2006, 86-92.
28. Zacks, J., B. Tversky and G. Iyer. Perceiving, Remembering, and Communicating Structure in Events. *Journal of Experimental Psychology: General*, *130* (1), 2001, 29-58.
29. Zacks, J.M. and B. Tversky. Event Structure in Perception and Conception. *Psychological Bulletin*, *127*, 2001, 3-21.