Examining Task Engagement in Sensor-Based Statistical Models of Human Interruptibility

James Fogarty, Amy J. Ko, Htet Htet Aung,
Elspeth Golden, Karen P. Tang, and Scott E. Hudson
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{ jfogarty, ajko, hha, egolden, kptang, scott.hudson }@cs.cmu.edu

ABSTRACT
The computer and communication systems that office workers currently use tend to interrupt at inappropriate times or unduly demand attention because they have no way to determine when an interruption is appropriate. Sensor-based statistical models of human interruptibility offer a potential solution to this problem. Prior work to examine such models has primarily reported results related to social engagement, but it seems that task engagement is also important. Using an approach developed in our prior work on sensor-based statistical models of human interruptibility, we examine task engagement by studying programmers working on a realistic programming task. After examining many potential sensors, we implement a system to log low-level input events in a development environment. We then automatically extract features from these low-level event logs and build a statistical model of interruptibility. By correctly identifying situations in which programmers are non-interruptible and minimizing cases where the model incorrectly estimates that a programmer is non-interruptible, we can support a reduction in costly interruptions while still allowing systems to convey notifications in a timely manner.

Author Keywords
Situationally appropriate interaction, managing human attention, sensor-based interfaces, context-aware computing, machine learning, interruptibility.

ACM Classification Keywords
H5.2. Information interfaces and presentation: User Interfaces; H1.2. Models and Principles: User/Machine Systems.

CHI 2005, April 2–7, 2005, Portland, Oregon, USA.

INTRODUCTION
Modern office workers increasingly find computing and communication systems at the core of their everyday work experience. At any given point in time, a person might be notified of the arrival of a new email, receive an instant message from a colleague, be reminded by a handheld computer of an upcoming appointment, receive a phone call on their office or mobile phone, and be involved in a face-to-face interaction with a colleague. Any one of these demands for attention can be addressed relatively easily, but simultaneous or repeated demands can quickly become disruptive. In a study of the perceptions of interruptions held by managers in a research organization, Hudson *et al.* found that some managers consider interruptions to be such a problem that they physically move away from their computers or even away from their offices in order to obtain uninterrupted working time [14]. The systems that office workers currently use tend to interrupt at inappropriate times or unduly demand attention in part because they have no way to determine when an interruption is appropriate. A colleague preparing to call a person cannot tell that the person is in the middle of debugging a complex program, and an email client about to announce the arrival of a new message cannot determine whether an obvious or subtle notification is currently more appropriate.

Sensor-based statistical models of human interruptibility offer a potential solution to this problem [8, 9, 11, 15]. For example, a phone system might automatically inform a caller that the callee appears to be busy, giving the caller the opportunity to consider the importance of the call and either leave a message or interrupt the callee. Prior work has explored a similar system that used a manually set do-not-disturb flag, but found that people often forgot to clear the flag when they became available, to the point that people considered the flag unreliable and ignored it [22]. Email clients and other systems could consider the importance of a notification relative to a person's current interruptibility, perhaps deferring or adjusting the salience of the notification when a person is busy.

Prior studies of the reliability of sensor-based statistical models of human interruptibility have primarily yielded results related to social engagement. Our own prior work used an experience sampling technique to collect self-reports of interruptibility from office workers in their normal work environments and built statistical models of their interruptibility based on sensors deployed in their offices. One of our primary findings was that a sensor to detect whether somebody in an office is talking is highly predictive of non-interruptibility [8, 9, 15]. Prior work by Horvitz and Apacible had office workers review recordings of themselves in their offices and provide labels of their own interruptibility. They found that electronic calendars, which capture planned social engagement, and perceptual systems capable of detecting ongoing social situations were both predictive [11].

While social engagement is clearly important and these results play an important role in deploying models, both our intuition and the literature suggest that task engagement is also important [14, 23, 25]. This paper presents work that more carefully examines task engagement in sensor-based statistical models of human interruptibility. Using an approach that we developed for our original studies of predicting human interruptibility, we examine how programmers respond to interruptions while they are programming and how statistical models can be used to predict their interruptibility. Within our approach, we develop a statistical model of the interruptibility of programmers that is based on low-level input events in a development environment. This model offers the potential to reduce costly interruptions at inappropriate times while still allowing appropriate notifications to be delivered in a timely manner. Furthermore, its performance is significantly better than the base performance typical of current systems that generally assume people are always interruptible.

Our approach is based on the fact that sensor development can be costly and time-consuming, regardless of whether a physical sensor is created in the real world or a software sensor is created in a digital world. Instead of a bottom-up approach to sensor development, wherein hardware and software sensing systems are developed, deployed, and evaluated, we present a top-down approach. In our top-down approach, we first study a problem to collect the data needed to make informed decisions about what sensors to implement, then develop and validate those sensors. The approach can be summarized as seven steps:

- Collect exploratory recordings of the environment into which sensors will be deployed.
- Collect a measure or estimate of the concept that will be predicted, which is human interruptibility in our work.
- Examine the collected recordings to develop ideas for what sensors may be predictive of the collected measure.
- Simulate the presence of those potential sensors in a systematic way from the collected recordings.
- Select sensors based on the utility of their simulated versions and the expected cost of their implementation.
- Implement the selected sensors.
- Validate the effectiveness of the implemented sensors in the deployment environment.

Our prior work has applied this approach to examining the interruptibility of office workers. In this paper, we use our approach to examine how programmers respond to interruptions when they are programming and what sensors might be used to predict their interruptibility.

In the next section, we further discuss our decision to examine task engagement and motivate our decision to study programmers. We then present our experimental setup and data collection mechanisms. This is followed by a discussion of the collected recordings and how those recordings influenced the direction of this work. We then present our simulated sensors and the results of our simulations. After brief comments on our implementation, we discuss our validation and present the resulting model. We then discuss our results and conclude.

TASK ENGAGEMENT AND PROGRAMMERS

Both our intuition and the literature suggest that capturing task engagement is important for creating reliable models of human interruptibility [14, 23, 25]. We also note that results from our prior work suggest that the task-related sensors we deployed in that work (software to detect the active application, other open applications, and the level of mouse and keyboard activity) could be significantly improved upon. Specifically, there are interesting differences in the classification errors that we observed with models of different types of office workers [9]. A model of two manager participants, selected because we expected their interruptibility to be heavily influenced by social engagement, correctly identified 90.6% of the interruptible manager observations and 81.0% of the non-interruptible manager observations. In contrast, a model of five researcher participants, all of whom did at least some programming and whom we expected to be less dominated by social engagement, also correctly identified 90.6% of interruptible researcher observations but only 60.9% of non-interruptible researcher observations. Some difference between the researcher participants and the manager participants resulted in models that were significantly less likely to detect that a researcher was non-interruptible ($\chi^2(1, 219) = 8.12$, $p < .01$), and it seems like this difference might be an inability of the deployed sensors to capture some types of task engagement.

Beyond our desire to further investigate this aspect of prior work, programmers have several other properties that make them good candidates for a study of task engagement in sensor-based statistical models of human interruptibility. Programming is a complex activity that places significant demands on working memory and other cognitive resources, and failures in working memory are known to

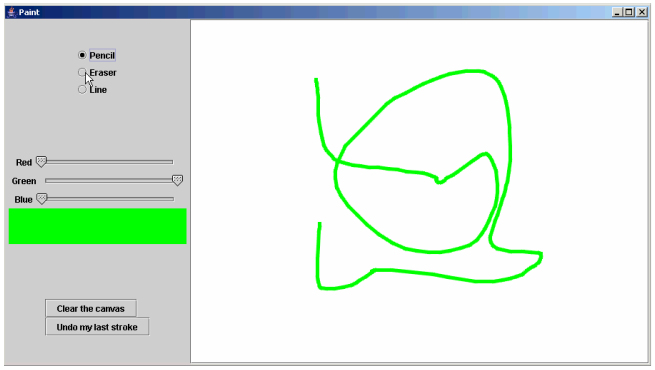

Figure 1. The *Paint* program that participants enhanced.

result in programming errors [1, 17]. Because interruptions increase the likelihood of such failures, we may be able to develop tools that use models of interruptibility to help reduce programming errors, perhaps by preventing interruptions or by noticing when programmers are interrupted at particularly non-interruptible points in their work and then helping them recover. Programmers also seem to be a good example of knowledge workers whose work involves significant interaction with computers, and so we are hopeful that results obtained with programmers will transfer to other computer-centric knowledge work. While results seem less likely to transfer to office workers who use computers relatively little, we are comfortable with this tradeoff because it seems like the advances offered by sensor-based statistical models of human interruptibility will initially be most relevant to computer-centric workers.

EXPERIMENTAL SETUP

In order to control the effects of social engagement and focus on task engagement, we studied programmers in a laboratory environment completing a realistic programming task while being subjected to interruptions. Participants worked in a small office that was free of other people or environmental distractions (other than the experimenter, who was available for questions about the task or the equipment but did not otherwise interfere). Participants worked in Eclipse 2.1.2, a modern development environment popular with Java programmers. Commercial screen capture software captured the entire screen at 12 frames per second in 24-bit color, with no noticeable impact on the computer's performance.

The *Paint* Program Primary Task

The *Paint* program, shown in Figure 1, is a 503-line program consisting of nine classes implemented in Java with the Swing toolkit. It provides basic paint support, allowing users to draw, erase, clear, and undo colored strokes on a white canvas. Participants were given the *Paint* program and allowed 70 minutes to address five requests. These requests were:

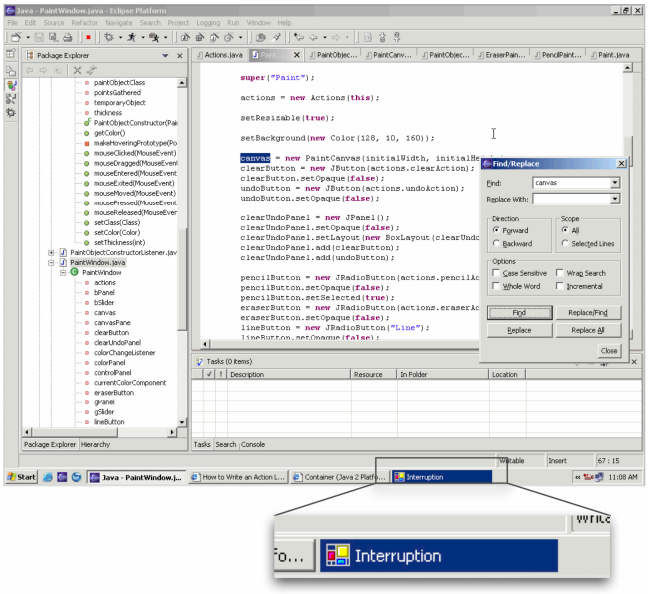

Figure 2. The Eclipse development environment, with a pending interruption flashing on the taskbar.

- "Users complained that scroll bars don't always appear after painting outside the canvas, but when they do appear, the canvas doesn't look right. Fix *Paint* so that (1) the scroll bars appear immediately when painting outside the visible canvas and (2) the canvas is correctly rendered when using the scroll bars to navigate the canvas."
- "Users complained that they can't select yellow. Fix *Paint* so that users can paint with the color yellow."
- "Users complained that the 'Undo my last stroke' button doesn't always work. Fix Paint so that the 'Undo my last stroke' button undoes the last stroke or clear of the canvas."
- "Users requested a line tool. There's a radio button for it, but it doesn't work yet. Create a line tool that allows users to draw a line between points. Users should be able to see the line while dragging."
- "Users requested control over the stroke thickness of the pencil, eraser, and line tools. Create a thickness slider with values from 1 to 50, which controls the thickness of the stroke for all tools."

Participants were given access to whatever resources they desired, including the Internet and the Java API documentation. They were also given full control over in what order to address the requests and how to manage their time. They were told they would be paid $10 for each successfully completed request.

Mental Arithmetic Interruptions

At random time intervals averaging approximately once every three minutes, an audio alert and the flashing taskbar

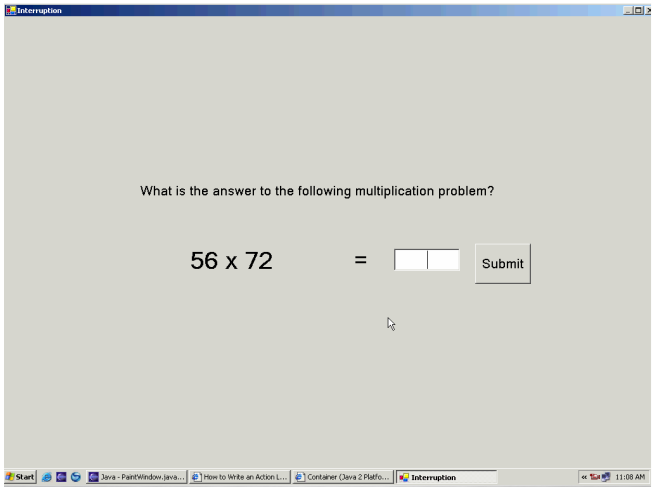

Figure 3. A mental arithmetic interruption. Note that it obscures the entire screen.

icon shown in Figure 2 notified participants of a pending interruption. Participants could choose whether to address it immediately or continue with their primary programming task until they wanted to address the interruption. In his work on techniques for coordinating interruptions, McFarlane refers to this approach as a negotiated solution, because a person is able to choose when they want to address an interruption. McFarlane found that this negotiated solution generally works best, as long as small differences in the time taken to begin addressing an interruption are not critical [21]. Robertson *et al.* have compared immediate and negotiated coordination of error messages when programming in a spreadsheet application, finding that task performance was significantly better with negotiated coordination [24]. This evidence that negotiated coordination is the best approach to interruptions in development environments informed our use of negotiated coordination, as we want our study to be based in how programmers handle normal interruptions. Value was associated with the interruptions by telling participants that they would lose $2 for any ignored or incorrectly answered interruption, but we did not enforce this penalty.

When the participant clicked on the notification, a two digit multiplication problem was presented, as in Figure 3. Participants were required to do the multiplication mentally, and were not allowed to use scratch paper or other programs. Because this task is known to place significant demands on working memory [20] and working memory failures are known to be a significant cause of programming errors [1, 17], this is an effective interruption of a programming task. To ensure the difficulty of the task, neither multiplicand had a 0 or 1 in its digits. To ensure participants understood the interruption mechanism and were practiced in mental arithmetic, several practice interruptions were given prior to the 70 minute primary task, during a 10 minute session of surfing web pages unrelated to programming.

EXPLORATORY DATA COLLECTION AND OVERVIEW

To collect our exploratory data and recordings, we recruited ten participants. Five were undergraduates majoring in computer science, four were graduate students in disciplines related to computing, and one was a graduate student in another field. Half of the subjects reported more than a year of industry programming experience, and the other half reported an average of less than two months of industry experience. This section reports on our exploratory analyses of the screen capture videos and how those analyses directed the work presented in the remainder of this paper.

Given the focus of this work on task engagement, we designed our primary programming task and the mental arithmetic interruption such that we could use a measure of interruptibility based in task performance, as opposed to more subjective measures like the self-reports used in our prior work [8, 9, 15] or the retrospective labeling used by Horvitz and Apacible [11]. However, prior to examining our exploratory recordings, it was unclear which of several potential measures was most appropriate.

After examining our exploratory data, we decided to measure interruptibility in terms of the difference between the time when the blinking taskbar notification was displayed, indicating that an interruption was pending, and the time that the participant acknowledged the interruption by clicking on the taskbar notification, which caused the multiplication dialog to take over the screen. We decided on this measure of interruptibility because the exploratory recordings repeatedly show participants finishing an edit or navigation before responding to the interruption. These behaviors are consistent with the notion that the participants were externalizing their working memory into the state of their development environment before addressing the pending interruption. Less abstractly, participants who were editing code tended to finish the edit before addressing the interruption, as opposed to responding to the interruption and then trying to resume the edit. Similarly, participants who were navigating to a particular location in the source code, such as a method or variable declaration, tended to finish the navigation, as opposed to addressing the interruption and then trying to remember to where they were navigating. This explanation held even for exceptionally long delays in addressing an interruption. For example, we inspected some delays of more than a minute that we initially thought resulted from missed or ignored notifications. We found that participants had pasted a large chunk of code shortly before the notification, and the long delay was due to the participant completing all planned modifications of the recently pasted code before attending to the interruption.

One of the more interesting potential measures of interruptibility that we considered but decided against is whether interruptions resulted in the introduction of actual errors into the *Paint* program. We did not pursue this measure of interruptibility because we found very few

Reading	
LINE	Highlights line(s) or moves cursor through a brief series of lines without editing
COMMENT	Edits comments in the code
PURUSE	Scrolls through code, but not to specific line of code, as if reading
BROWSE	Expands nodes and/or scrolls in the package explorer, but not to a specific object
HOVER	Interacts with the system, but hovers the cursor over a specific line of code or explorer object
IDLE	Does not interact with mouse or keyboard for more than 2 seconds

Coding	
EDIT	Edits program code, including any cursor movements or line selections

Code Navigation	
FIND	Expands nodes and/or scrolls in the package explorer to a specific object
GOTO	Scrolls to a specific line of code
METHOD	Opens a method or variable from the package explorer
SEARCH	Places the cursor in a text field to searching or replace

Interface Navigation	
UI	Searches menus, context menus, or toolbars for commands for more than 1 second
WAIT	Is waiting for a progress bar or hour glass cursor

Task Switching	
ACTIVATE	Makes the Eclipse window active
OUTSIDE	Performs actions outside the Eclipse environment and their program
RUN	Executes the program with CTRL-F11, the Run button, or the menu item
TEST	Interacts with their program
VIEW	Switches Eclipse perspectives, or closes or open a source file

Debugging	
FIND	Expands nodes and/or scrolls in the package explorer to a specific object
GOTO	Scrolls to a specific line of code

Figure 4. The set of 21 simulated sensors, grouped into 6 categories.

instances of an error being introduced as a result of an interruption. We believe this is because the negotiated coordination of the interruptions allowed participants to carefully externalize their working memory in the development environment, as just discussed, and note that Gillie and Broadbent found similar results [10]. We believe that more programming errors would have resulted if the interruptions had been presented without warning or delay, but decided against this approach in order to remain closer to the types of interruptions that programmers experience in their normal programming environments.

SIMULATED SENSORS

Based on the major activities we observed in the exploratory recordings and our decision to measure interruptibility as the time for a participant to respond to an interruption notification, we developed a set of 21 simulated sensors in 6 categories, shown in Figure 4. These simulated sensors were chosen because they occurred with reasonable frequency in the exploratory recordings, because they seem like they might relate to interruptibility, and because they might reasonably be implemented. While the specifics of some of these sensors, such as the difference between *PERUSE* and *GOTO*, might be rather difficult to implement, we included them because knowing that such a sensor would be predictive could possibly justify the effort needed to develop it.

Working from a specification of when to mark the beginning and end of activation for each of these simulated sensors, we simulated their output for the minute preceding each notification of a pending mental arithmetic interruption. We then created features to capture the frequency, duration, and recency of simulated sensor activation and built statistical models from these features, attempting to predict the interruptibility of the participants. We will not present a detailed analysis of these simulated sensors, leaving such a presentation for the results obtained with our implemented sensors. Instead, we now comment on the results of these analyses that influenced our choice of sensor implementation.

While we had expected the *EDIT* sensor to be useful, we were surprised to find it was the only simulated sensor to emerge as predictive. This might be because the other activities for which we created simulated sensors do not have the same working memory requirements as editing, and so therefore do not result in delays when responding to an interruption. It might also be that they impose working memory requirements, but do not occur often enough in our collected data to emerge as predictive of interruptibility. In any case, this result led us to focus on implementing a sensor to detect the frequency, duration, and recency of low-level input events. On the other hand, if simulated sensors like *PERUSE*, which is based in the activities a participant is performing over a period of time, had emerged as predictive, we might have instead chosen to implement a sensor that analyzed sequences of input events to detect different patterns.

IMPLEMENTATION

We implemented our sensor by developing an Eclipse plug-in that subscribes to every system event generated by widgets in the Eclipse development environment. Because Eclipse is implemented in the Standard Widget Toolkit (SWT), our plug-in uses the SWT to start from each top-level SWT window and recursively descend its widget hierarchy, adding appropriate SWT event listeners to each widget encountered. These event listeners log the appropriate parameters of each low-level event. This recursive search is executed twice per second, such that newly created widgets and dialogs are detected and logged.

Beyond logging the basic parameters of each event, the plug-in also logs some additional information more specific to the programming task. For appropriate events, our

plug-in examines the source code currently visible in the editor, logging which methods of which classes are visible and how many lines of code from each of those methods are visible. This allows events to be associated with classes and methods, rather than just characters or screen coordinates.

VALIDATION DATA COLLECTION

In order to examine the effectiveness of our implemented sensor, we recruited twenty additional participants. Eight were undergraduates majoring in computer science, six were undergraduates majoring in related fields, two had bachelor's degrees in other fields and several years of industry programming experience, two were graduate students in computer science, and two were graduate students in related fields. Thirteen participants reported more than a year of industry programming experience, and the other seven reported an average of less than two months of industry experience. We collected a total of 475 response time observations from these participants.

In order to apply a classifier, we clustered participant response times using the Expectation Maximization algorithm [6] as implemented in Weka, an open source machine learning toolkit [26]. Given a set of values and a number of clusters to produce, the algorithm computes the means and standard deviations of the normal distributions most likely to have generated the given values. We examined the algorithm's output for two, three, and four clusters and decided to proceed by analyzing the data in three clusters, for reasons we will discuss later in this section.

The first cluster, which we will refer to as *interruptible,* represents an immediate response to an interruption notification and contains 278 response time observations with a mean of 2.281 seconds and a standard deviation of 752 milliseconds. The second cluster, which we will refer to as *engaged,* represents a short delay from notification to response and contains 143 response time observations with a mean of 6.917 seconds and a standard deviation of 3.434 seconds. The final cluster, which we will refer to as *deeply engaged*, represents a long delay from notification to response and contains 54 response time observations with a mean of 43.065 seconds and a standard deviation of 37.399 seconds. Each pair of clusters is significantly different (*interruptible* vs. *engaged*: $t(419) = 21.55$, $p < .001$, *interruptible* vs. *deeply engaged*: $t(330) = 18.28$, $p < .001$, *engaged* vs. *deeply engaged*: $t(195) = 11.48$, $p < .001$).

Note that the very small deviation in response time for observations in the *interruptible* cluster was one of the reasons we decided to use three clusters to analyze this data, as we believe it indicates a more realistic partitioning of the data. When using only two clusters, observations in our *interruptible* and *engaged* clusters were grouped together into a single cluster with a mean of 2.946 seconds, but the small deviation of response times for observations in the *interruptible* cluster gave this combined set of observations a standard deviation of 1.529 seconds. The larger standard deviation meant that some entries from our *engaged* cluster instead appeared in the second cluster generated, which had a mean of 27.116 seconds and a standard deviation of 31.601 seconds. Examining this data, we felt that three clusters represented a more appropriate division of the data, and found that moving to four clusters appeared to offer no improvement.

ANALYSIS

This section first presents our extraction of a set of features from the raw event logs collected by our sensor implementation. We then discuss our method for selecting a useful subset of these features and present the accuracy of a statistical model constructed from this useful subset. Finally, we present some of the most predictive features in the model, so that the community might include such features in their systems.

Feature Extraction

Unprocessed sensor data is typically inadequate for creating sensor-based statistical models, for a variety of reasons. One reason is that observations of the value that a model will predict (the interruptibility of a programmer in our work) tend to be relatively sparse compared to the abundance of sensor data. This makes it important to consider recent values of sensors, rather than training models from only the sensor data available at the exact moment of each observation of the value that is to be predicted. Another reason can be found in continuous sensor data, such as the number of keystrokes in the last minute. Because statistical models are based on extracting correlations between input features and the value to be predicted, better models typically result if such continuous values are discretized, a process of grouping ranges of values into discrete bins, prior to model construction. This allows the determination, for example, that it is not important whether two or three keystrokes have occurred in the last five seconds, rather it is only important that more than zero have occurred.

In our prior work and in our analysis of our simulated sensors, we used manually-developed scripts to extract features based on the frequency, recency, and duration of events in the raw sensor logs. For our evaluation analysis, however, we automatically extracted features from our raw sensor data using `AmIBusy`, a system that we are creating to support the development and deployment of sensor-based statistical models of human interruptibility. `AmIBusy` works by recursively applying sequences of operators to sensor data. For example, one operator can get all of the sensor readings reported as XML strings by a sensor in the 60 seconds prior to an interruptibility observation, another operator can then extract the value of an attribute from the XML for each reading, a third can convert the value from a string to a double, and a fourth can discretize the double.

While AmIBusy and automated feature extraction are not the topic of this paper, it is important to point out that the features used by our models were automatically generated, because we want to make clear that the model construction process we use can be fully automated. We do not believe that sensor-based statistical models of interruptibility should be considered static entities in deployed systems, but rather believe that they should be continuously and automatically refined. The automated extraction of appropriate features from low-level sensor data is an important part of making this feasible. AmIBusy enables this, and the results we present here demonstrate the effectiveness of such an approach.

Feature Selection

Not all of the features that are automatically extracted from raw sensor data are appropriate for use in a statistical model. For some features this is because they simply do not correlate with interruptibility and so are not predictive. Other features are inappropriate for use in a statistical model due to a phenomenon known as over-fitting. Because it is computationally intractable to examine the utility of every potential combination of features, a two-stage heuristic search is used to select an appropriate subset of the generated features.

This heuristic search starts by selecting the individual features that are most correlated with interruptibility, using three common but distinct measures of correlation (information gain, gain ratio, and symmetrical uncertainty). For each measure, AmIBusy selects the 250 features most correlated with interruptibility. Additional features are then selected using Yu and Liu's fast-correlation based filter technique [27] with each measure of correlation. This technique selects a small number of features that are not in the top 250 but have some predictive value that is distinct from the top 250 features. These correlation-based techniques can be applied quickly, and the union of features selected is used as input to the second stage of the search.

The second stage of the search is a wrapper-based feature selection, which examines the predictiveness of different combinations of potential features. In a wrapper-based feature selection, the algorithm starts with an empty set of features and slowly adds or removes features, examining the effect of these changes on the accuracy of the model being constructed. When no change to the candidate subset results in an improvement, the feature selection terminates [18]. While this optimization can select the best feature subset, it is computationally expensive. It is this expense that motivates us to first reduce the size of the search by using correlation-based techniques.

Resulting Model

Using the feature generation and selection mechanisms just discussed, we constructed a statistical model of the interruptibility of the programmers in our experiment. This model differentiates between *interruptible* observations and

		Naïve Bayes Model	
		Interruptible	Engaged or Deeply Engaged
Response Time	Interruptible	208 43.8%	70 14.7%
	Engaged or Deeply Engaged	64 13.5%	133 28.0%
		Accuracy: 71.8% Base: 58.5%	

Figure 5. Accuracy of a model to distinguish interruptible situations from other situations (engaged or deeply engaged).

observations in one of the other two clusters, *engaged* and *deeply engaged*. Figure 5 shows the accuracy of this model, presented in a form known as a confusion matrix. This result was obtained using a standard ten-fold cross-validation, wherein ten trials of model construction are executed, each using 90% of the data for training and the remaining 10% to evaluate the model in that trial. The values reported are the sums from all ten trials. The rows represent the actual number of response time observations in each group, and the columns represent the estimates made by our sensor-based statistical model. The unshaded diagonal, therefore, represents the cases there the model correctly predicted whether the participant was interruptible. Conversely, the shaded diagonal represents the cases where the model was incorrect.

With an overall accuracy of 71.8%, this model is significantly more accurate than the base performance of 58.5% typical of current systems, which generally assume that people are always interruptible ($\chi^2(1, 950) = 18.4$, $p < .001$). Built with a naïve Bayes classifier [7, 19], this model is based on 23 automatically generated features. Considering the expected difficulty of building a reliable model of the interruptibility of programmers from low-level input events, we consider this a strong result. The difference between the models dominated by social engagement in our prior work and the task-related models built in this work mean that we should be careful not to directly compare their accuracy. However, it would be unreasonable to expect perfect performance, as our prior work has shown that human observers of office workers can only estimate the interruptibility of those office workers with an accuracy of 76.9% [8]. In using sensor-based statistical models of human interruptibility, whether in social environments or task-based environments, it will be important for applications to negotiate entry into interruptions, rather than treating an interruptibility estimate as if it provides absolute guidance.

We also note that applications can threshold the output of this model based on their own needs. The confusion matrix in Figure 5 was computed by labeling a situation as interruptible if the probability of being interruptible output by the model was .617 or greater (this probability was chosen to maximize the overall accuracy). At this

threshold, the model correctly detects 74.8% of interruptible situations and 67.5% of non-interruptible situations. This tradeoff might be appropriate for some applications, but other applications might prefer higher detection of interruptible situations. Choosing a different tradeoff does not necessarily imply a large sacrifice in accuracy, as using our model with an interruptible threshold of .372 results in an accuracy of 71.6%. At this threshold, the model correctly detects 90.0% of interruptible situations, while still detecting 45.7% of non-interruptible situations. We will not introduce ROC curves in the space available here, but note that our model has an A' value of .776, significantly better than chance ($Z = 18.7$, $p < .001$).

Selected Features

To provide more insight into our model, Figure 6 presents an ordered list of its seven most predictive features. This ordering was determined using an accuracy-based forward selection through the features in the model. So the first feature is the single feature that results in the highest accuracy. Once this first feature has been chosen, the second feature is the best addition to the model, and so on. The accuracy column on the extreme left presents the cumulative effect of these features on the accuracy of the model. After adding the seventh feature, the model has an accuracy of 70.5%. The additional 16 features, not shown here, improve the accuracy of the model to 71.8%.

For each feature, Figure 6 shows how different values of the feature influence the model's estimate of whether a person is interruptible or engaged. Consider the first feature, how many ExtendedModify events (which are generated whenever the text of a file is modified) occurred in the 15 seconds prior to an interruption. If we look only at the data for which no ExtendedModify events occurred, there are 132 situations where the programmer was interruptible and 86 non-interruptible situations. These 218 situations represent 45.9% of our total data, and our participants were interruptible for 60.6% of these situations. Accounting for rounding in our presentation, the difference between this likelihood of being interruptible and the 58.5% likelihood in the full data set is +2.0%. This shows it was common for no ExtendedModify events to have occurred in the 15 seconds before an interruption (45.9% of the data), but this meant relatively little in terms interruptibility (the +2.0%). On the other hand, when more than one ExtendedModify event had occurred in the past 15 seconds, participants were much less likely to be interruptible (a difference of -15.6%). The magnitude of this difference, together with the fact that it occurred in 25.5% of the data, is a big part of why this feature was selected as most predictive. This is also consistent with our simulated sensors, which found editing to be the primary indicator.

Because considering a programmer non-interruptible if they have recently edited text is a more satisfying baseline than assuming a programmer is always interruptible, we note that our full model performs significantly better than a model based on this single feature (*comparing accuracy*: $\chi^2(1, 950) = 10.1$, $p \approx .001$, *comparing area under the ROC curve: .776 vs .577*, $Z = 5.7$, $p < .001$).

Accuracy	Value	Num Int	Num Non	% of Data	% Int	Change
58.5%	**No Features**					
62.1%	**ExtendedModify Event in Past 15 Seconds**					
	No Event	132	86	45.9%	60.6%	+2.0%
	1 Event	94	42	28.6%	69.1%	+10.6%
	> 1 Event	52	69	25.5%	43.0%	-15.6%
64.6%	**Application Focus Change in Past 5 Seconds**					
	No Change	221	151	78.3%	59.4%	+0.9%
	Eclipse Gained	36	12	10.1%	75.0%	+16.5%
	Eclipse Lost	21	34	11.6%	38.2%	-20.3%
67.2%	**Switch Between Modifying PaintWindow.java and Modifying Another File in the Past 15 Seconds**					
	< 2 Files Modified	123	78	42.3%	61.2%	+2.7%
	2 Files Modified	138	116	53.5%	54.3%	-4.2%
	3 Files Modified	17	2	4.0%	89.5%	+30.9%
68.6%	**Most Common Key Event in Past 60 Seconds**					
	No Key	150	101	52.8%	59.8%	+1.2%
	Key Pressed	14	13	5.7%	51.9%	-6.7%
	Key Released	65	26	19.2%	71.4%	+12.9%
	Key Traversed	49	57	22.3%	46.2%	-12.3%
69.1%	**Recently Stopped Typing (Last Key Event 15 to 20 Seconds Ago)**					
	< 15 or > 20	267	197	97.7%	57.5%	-1.0%
	15 to 20	11	0	2.3%	100%	+41.5%
69.5%	**Tree Event in Past 5 Seconds**					
	No Events	228	179	85.7%	56.0%	-2.5%
	1 Event	43	17	12.6%	71.7%	+13.1%
	2 Events	6	0	1.3%	100%	+41.5%
70.5%	**Control Resize Event in Past 5 Seconds**					
	No Resize	265	178	93.3%	59.8%	1.3%
	Controls Resized	13	19	6.7%	40.6%	-17.9%

Figure 6. Seven most predictive features from the model.

The additional features show that programmers were less interruptible if they had recently switched away from Eclipse, perhaps because they were in the middle of testing the *Paint* application or searching through documentation. Subjects were also less interruptible if modifying two files, but the difficulty of modifying three files in 15 seconds makes it likely that those 19 cases are an artifact of the task, possibly due to a need for making a single small paste into multiple files. The next feature shows that programmers were less interruptible when generating KeyTraversed events, which the SWT sends when the arrows or other keys are used to move the input cursor. The fourth feature, indicating that subjects were interruptible when their last keyboard event occurred between 15 and 20 seconds in the past, might be related to task completion, but it occurs too infrequently to warrant much confidence. Programmers were also more interruptible when they were interacting with tree controls. This is consistent with our simulated sensor results, as a programmer interacting with a tree control cannot be actively editing code. The seventh feature shows that programmers were less interruptible when they had recently resized part of the Eclipse environment, which may be related to maximizing one of the child windows in the development environment to focus on its contents.

RELATED WORK

Field studies of interruptions and how people perceive their interruptibility have informed our work, but do not directly inform the deployment of sensor-based statistical models of human interruptibility. As mentioned in our introduction, Hudson *et al.* studied the perceptions of interruptions held by managers in a research organization, finding that some managers consider interruptions to be such a problem that they physically move away from their computers or even away from their offices in order to obtain uninterrupted working time [14]. Perlow found a self-perpetuating cycle of interruptions in the workplace, wherein workers in danger of missing a deadline interrupt other workers with requests, which then causes the interrupted workers to fall behind in their own work, leading them to then interrupt others [23].

Field studies that more directly inform sensor-based statistical models of human interruptibility have typically reported findings primarily related to social engagement. In our own prior work, we have used the approach presented in this paper to study the interruptibility of office workers in their normal working environments [8, 9, 15]. In that work, we measured interruptibility using an experience sampling technique, prompting workers to report their interruptibility at random intervals approximately once per hour. We collected data for several weeks from each participant, and showed that real sensors could support models of their interruptibility with accuracies as good as or better than human observers. These results were largely dominated by social engagement, helping to motivate the work presented in this paper. Horvitz and Apacible also directly studied interruptibility, asking workers to retrospectively review several hours of collected recordings to provide labels of their interruptibility, then examining models of these labels based on system events, perceptual analyses of audio and video streams, and electronic calendar entries [11]. They do not explicitly differentiate between sensors related to task engagement and social engagement, but the perceptual systems and electronic calendar analyses on which their discussion is focused seem to be primarily related to social engagement.

Other work has studied interruptibility in laboratory tasks, but without the goal of enabling sensor-based statistical models of human interruptibility. For example, Czerwinski *et al.* examined interruptions by instant message notifications during some relatively simple list-browsing and office software tasks, finding that even ignored notifications can be disruptive [4, 5]. Gillie and Broadbent studied resumption of a task after different types of interruptions, also finding that the externalization of working memory into the state of the task meant that very few errors resulted from interruptions [10]. McFarlane points out that models of interruptibility can be used as part of a mediated approach to coordinating interruptions, but his studies of interruptions and task performance compare strategies for coordinating interruptions, rather than informing the development of sensor-based statistical models of human interruptibility [21]. Robertson *et al.* studied interruption coordination specifically in the context of spreadsheet programming, finding that negotiated coordination lead to better task performance, but their work does not inform the development of statistical models of the interruptibility of programmers [24].

A variety of systems have explored concepts related to interruptibility. The Priorities system, by Horvitz *et al.*, considers patterns of prior device access to reason about when a person is likely to be available on a given device, such as a personal computer or a mobile phone, and can consider the apparent importance of a message in deciding whether to forward it to a mobile device [12]. The Coordinate system, also by Horvitz *et al.*, adds perceptual sensors based on audio and video streams, together with analyses of electronic calendar entries, to reason about the presence and availability of people [13]. Begole *et al.* analyzed logs of presence in their Awarenex system and developed a method to automatically extract temporal patterns, such as recurring meetings [2, 3]. Kern and Schiele examined the ability of wearable sensors to detect different contexts and activities that they argue relate to interruptibility [16]. A major difference between our work and these systems is our use of an evaluation based on an explicit measure of interruptibility. Evaluations of other systems typically examine the ability of a system to recognize particular contexts, but do not explicitly evaluate how those contexts actually relate to interruptibility.

DISCUSSION AND CONCLUSION

We have presented our work to more carefully explore task engagement in sensor-based statistical models of human interruptibility by studying the interruption of programmers working on a realistic programming task. This work contributes an evaluation of a model based on low-level system events in a development environment, finding that it can distinguish situations where a programmer is interruptible from other situations with an accuracy of 71.8%, significantly better than the base accuracy of 58.5% accuracy typical of current systems that generally assume that a programmer is always interruptible and significantly better than a model that assumes programmers are non-interruptible if they recently edited their code.

Beyond the specifics of the models created in this work, we contribute an explicit presentation and application of our approach to developing sensor-based statistical models. Because sensor development can often be costly and time-consuming, our approach is based on collecting recordings from an environment into which sensors will be deployed and then using the recordings to simulate the presence of sensors that might be predictive. The information obtained from these simulated sensors allows informed decisions to be made about which sensors to implement, thus making it more likely that the resulting system will be successful.

Moving forward, there are several lines of future work we intend to pursue. The models presented here offer to reduce the costs of inappropriate interruptions experienced by programmers, and we are interested how to best deploy such models into programming environments. We are also interested in expanding these results beyond programmers, perhaps by studying task engagement with different types of office workers or by building a sensor that logs low-level input events in the entire system, rather than just in the development environment. Our approach to developing sensor-based statistical models of human interruptibility has yielded strong results in both our prior work and this work, but we remain interested in the possibility of refining or improving it. We also intend to integrate the results of this work and the results of our prior work, with the goal of developing more accurate models of human interruptibility.

ACKNOWLEDGMENTS

We would like to thank Santosh Mathan for his work in an early stage of this research. We would also like to thank all of the contributors to Weka. This work was funded in part by DARPA, by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298, by an NDSEG fellowship, by an AT&T Labs fellowship, and by the National Science Foundation under grants CCR-03244770, IIS-0329090, IIS-0121560, and IIS-0325351. The views and conclusions contained in this document are those of the authors.

REFERENCES

1. Anderson, J.R. and Jeffries, R. (1985) Novice LISP Errors: Undetected Losses of Information from Working Memory. *Human-Computer Interaction*, *1* (2). 107-131.
2. Begole, J.B., Tang, J.C. and Hill, R. (2003) Rhythm Modeling, Visualizations, and Applications. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2003)*, 11-20.
3. Begole, J.B., Tang, J.C., Smith, R.B. and Yankelovich, N. (2002) Work Rhythms: Analyzing Visualizations of Awareness Histories of Distributed Groups. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2002)*, 334-343.
4. Cutrell, E., Czerwinski, M. and Horvitz, E. (2001) Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. *Proceedings of the IFIP Conference on Human-Computer Interaction (INTERACT 2001)*, 263-269.
5. Czerwinski, M., Cutrell, E. and Horvitz, E. (2000) Instant Messaging and Interruptions: Influence of Task Type on Performance. *Proceedings of the Australian Conference on Computer-Human Interaction (OZCHI 2000)*, 356-361.
6. Dempster, A.P., Laird, N.M. and Rubin, D.B. (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, *39* (1). 1-38.
7. Duda, R.O. and Hart, P.E. (1973) *Pattern Classification and Scene Analysis*. John Wiley and Sons.
8. Fogarty, J., Hudson, S., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J. and Yang, J. (2004) Predicting Human Interruptibility with Sensors. *To Appear, ACM Transactions on Computer-Human Interaction (TOCHI)*.
9. Fogarty, J., Hudson, S. and Lai, J. (2004) Examining the Robustness of Sensor-Based Statistical Models of Human Interruptibility. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*, 207-214.
10. Gillie, T. and Broadbent, D. (1989) What Makes Interruptions Disruptive? A Study of Length, Similarity, and Complexity. *Psychological Research*, *50*. 243-250.
11. Horvitz, E. and Apacible, J. (2003) Learning and Reasoning about Interruption. *Proceedings of the International Conference on Multimodal Interfaces (ICMI 2003)*, 20-27.
12. Horvitz, E., Jacobs, A. and Hovel, D. (1999) Attention-Sensitive Alerting. *Proceeding of the Conference on Uncertainty and Artificial Intelligence (UAI 1999)*, 305-313.
13. Horvitz, E., Koch, P., Kadie, C.M. and Jacobs, A. (2002) Coordinate: Probabilistic Forecasting of Presence and Availability. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, 224-233.
14. Hudson, J.M., Christensen, J., Kellogg, W.A. and Erickson, T. (2002) "I'd be overwhelmed, but it's just one more thing to do": Availability and Interruption in Research Management. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2002)*, 97-104.
15. Hudson, S., Fogarty, J., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J. and Yang, J. (2003) Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*, 257-264.
16. Kern, N. and Schiele, B. (2003) Context-Aware Notification for Wearable Computing. *Proceedings of the IEEE International Symposium on Wearable Computing (ISWC 2003)*.
17. Ko, A.J. and Myers, B. (2004) A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems. *To Appear, Journal of Visual Languages and Computing*.
18. Kohavi, R. and John, G.H. (1997) Wrappers for Feature Subset Selection. *Artificial Intelligence*, *97* (1-2). 273-324.
19. Langley, P. and Sage, S. (1994) Induction of Selected Bayesian Classifiers. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 1994)*, 399-406.
20. Lemaire, P., Abdi, H. and Faylo, M. (1996) The Role of Working Memory Resources in Simple Cognitive Arithmetic. *European Journal of Cognitive Psychology*, *8* (1). 73-103.
21. McFarlane, D.C. (2002) Comparison of Four Primary Methods for Coordinating the Interruption of People in Human-Computer Interaction. *Human-Computer Interaction*, *17* (1). 63-139.
22. Milewski, A.E. and Smith, T.M. (2000) Providing Presence Cues to Telephone Users. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2000)*, 89-96.
23. Perlow, L.A. (1999) The Time Famine: Toward a Sociology of Work Time. *Administrative Science Quarterly*, *44* (1). 57-81.
24. Robertson, T.J., Prabhakararao, S., Burnett, M., Cook, C., Ruthruff, J.R., Beckwith, L. and Phalgune, A. Impact of Interruption Style on End-User Debugging. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*, ACM Press, 2004, 287-294.
25. Seshadri, S. and Shapira, Z. (2001) Managerial Allocation of Time and Effort: The Effects of Interruptions. *Management Science*, *47* (5). 647-662.
26. Witten, I.H. and Frank, E. (1999) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.
27. Yu, L. and Liu, H. (2003) Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. *The International Conference on Machine Learning (ICML 2003)*, 856-863.