Constructing and Evaluating Sensor-Based Statistical Models of Human Interruptibility

James Anthony Fogarty

January 2006
CMU-HCII-06-100

Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Scott E. Hudson (Chair)
Christopher G. Atkeson
Robert E. Kraut
Eric J. Horvitz, Microsoft Research

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This research was supported in part by the National Science Foundation under grants CCR-03244770, IIS-0121560, IIS-0205219, IIS-0325351, IIS-0329090, IIS-9800597, and IIS-9980013, by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298, by the author's internship at IBM Research, by the author's National Science Foundation Graduate Research Fellowship, by a collaborator's National Defense Science and Engineering Graduate Fellowship, and by a collaborator's AT&T Labs Fellowship. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under contract No. NBCHD030010.

Keywords: Human interruptibility, interruptions, office workers, sensor-based statistical models, Wizard of Oz sensor development, `Subtle`, AmIBusy Prompter, Whistle.

Abstract

While people can typically make a rapid assessment of another person's interruptibility, current systems generally have no way to consider whether an interruption is appropriate. Systems therefore tend to interrupt at inappropriate times or unduly demand attention. Sensor-based statistical models of human interruptibility are one approach to addressing this problem. In a series of studies, we examine the feasibility and robustness of sensor-based statistical models of human interruptibility, creating models that perform better than human observers. We then present a tool to enable non-expert development of applications that use sensor-based statistical models of human situations.

Our first study collects audio and video recordings in the normal work environments of several office workers. We measure their interruptibility by collecting interruptibility self-reports via experience sampling. We then use a Wizard of Oz method to examine the recordings and simulate many potential sensors. Building statistical models from these simulated sensors, we are able to evaluate potential sensors without actually building them. In our second study, human observers view the recordings and estimate the interruptibility of the office workers. Statistical models based on our simulated sensors perform better than these human observers. Our third study examines the robustness of this result by implementing and deploying real sensors with a more diverse set of office workers. While different sensors are more predictive for different types of office workers, even a general model performs better than the human observers. Because these first three studies are dominated by social engagement, our fourth study explicitly examines task engagement. We show that low-level programming environment events can be used to model when a programmer will choose to defer an interruption.

We then develop `Subtle`, a tool to enable further research into how human computer interaction can best benefit from sensor-based statistical models of human situations. With an extensible sensing library, fully-automated iterative feature generation, and support for model deployment, `Subtle` enables non-expert development of applications that use sensor-based statistical models of human situations. `Subtle` allows human computer interaction researchers to focus on compelling applications and datasets, rather than the difficulties of collecting appropriate sensor data and learning statistical models.

Finally, we present a summary of contributions and plans for future work.

Acknowledgements

Many people have touched my life, my education, and my work. While I hope they already know of the impact they have had, I want to explicitly thank some of the people to whom I am most grateful.

Any success I have as a researcher should be largely attributed to those who have guided and participated in my work. As my advisor, Scott Hudson has taught me to choose compelling problems, given me the freedom to find my own answers, and shared a chuckle when it became obvious why my initial answer was so completely wrong. Beyond showing me how to conduct good research, Scott has taught me many life lessons about being a good researcher. The other members of my committee have also provided immeasurable guidance. Christopher Atkeson has shared a practical outlook on how to work past various distractions to focus on the question at hand. Eric Horvitz has helped to pique my interest in pursuing hard research problems that also have the potential for a real impact on how we live. Robert Kraut has sought and helped to provide clarity in my work, showing me how to break down and pursue complex problems. I am also indebted to many co-authors and colleagues in my graduate career, including Gregory Abowd, Carolyn Au, Htet Htet Aung, Daniel Avrahami, Ryan Baker, Jim Christensen, Laura Dabbish, Jodi Forlizzi, Darren Gergle, Elspeth Golden, Jason Hong, Pedram Keyani, Sara Kiesler, Andrew Ko, Jennifer Lai, Johnny Lee, Joonhwan Lee, Brad Myers, Jack Mostow, Judith Olson, David Steck, Karen Tang, Sebastian Thrun, and Jie Yang.

Any success I have as a mentor or an educator should be credited to the people who shaped my own educational experience. In their role as undergraduate research advisors, John Carroll and Mary Beth Rosson helped to develop my interest in human-computer interaction, as did all of the students and staff in their research group. Sandra Birch was an asset to every Computer Science undergraduate at Virginia Tech, and Sallie Henry provided valuable advice throughout my time there. In my interaction with the University Honors Program, Jack Dudley and Barbara Cowles helped to shape my expectations of scholarship and society. I am also thankful for the many teachers who contributed to my education in my childhood and my adolescence, especially Mrs. Wells. She always said that I would eventually appreciate all the things she got me involved in, and of course she was right.

Any success I have in life is largely due to my parents, James and Kathleen. Whether it was camping, soccer, or schoolwork, my parents have always taken an interest and made time for me. My father accidentally taught me the value of holding onto monkey bars, and my mother taught me not to wander off in Kmart. While I am eternally thankful to have been raised in such a loving and supportive environment, I am even more grateful to now know my parents as friends. My relationship with my younger brother, Tom, includes quite a bit of sibling rivalry, but we have always stood up for each other when it mattered. If I have shown him anything, it is that playing tag with darts is not exactly a brilliant idea.

Finally, any success I have in love is due entirely to Rebecca. I can wear down anybody's patience, but she never seems to tire of me. She has taught me to share, and she calls my bluffs. She makes me laugh, and I smile at her when she's not looking. We giggle and fight like toddlers, and I am an infinitely better person for knowing her. When I finally asked her to marry me, it caught her completely by surprise. But I am so happy that she remembered to say yes.

Table of Contents

List of Figures

1 Introduction

2 Related Work

3 A Wizard of Oz Approach To Top-Down Sensor Development

4 A Wizard of Oz Study of Office Worker Interruptibility

5 A Study of Human Observers Estimating Office Worker Interruptibility

6 A Robustness Study with Real Sensors and Diverse Office Workers

7 A Study of Task Engagement in a Natural Programming Problem

8 Toolkit Support for Sensor-Based Statistical Models of Human Situations

9 Conclusion and Future Work

Chapter 1

Introduction

1.1 Motivation

People have developed a variety of conventions that defined what behavior is socially appropriate in different situations (Barker 1968). In office working environments, social conventions dictate when it is appropriate for one person to interrupt another. These conventions, together with the reaction of the person who has been interrupted, allow an evaluation of whether or not an interruption is appropriate. Social conventions surrounding interruptions also allow the development of an *a priori* expectation of whether or not an interruption would be appropriate (Hatch 1987). For example, a person seeking a colleague's attention is normally able to glance in an open office door and quickly assess the colleague's current interruptibility.

Current computer and communication systems are largely unaware of the social conventions defining appropriate behavior, the social situations that surround the use of an application, and the impact that an application's actions have on social situations. For example, mobile phones ring while their owners are in meetings and laptop computers regularly interrupt presentations to announce that a new email has arrived or that the laptop's battery is fully charged. Current computer and communication systems frequently create socially awkward interruptions or unduly demand attention because they have no way to determine whether it is appropriate to interrupt. Because they have no model of interruptions, it is impossible for these systems to develop an *a priori* expectation of the impact their interruptions will have on users and the social situations surrounding an application.

The general failure of current systems to consider our interruptibility can overwhelm modern office workers. 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 so disruptive that they physically move away from their computers or even away from their offices in order to obtain uninterrupted working time (Hudson et al. 2002). Perlow found that interruptions in the workplace can create a self-perpetuating cycle, wherein workers in danger of missing a deadline interrupt other workers with urgent requests, which then causes the interrupted workers to fall behind in their own work, leading them to interrupt still more workers (Perlow 1999).

People who design or use current computer and communication systems can generally adopt one of two strategies for managing the damage caused by inappropriate interruptions. One strategy is to avoid building or using proactive systems, forcing systems to be silent and wait passively until a user initiates interaction. This approach is reasonable for many applications in a desktop environment, but many applications in intelligent spaces or other ubiquitous computing environments could benefit from a system being able to initiate interactions (Horvitz 1999). Even common desktop applications sometimes need to initiate an interruption, as in the case of spreadsheet applications that detect an error in a formula and need to notify the user (Robertson et al. 2004). A second strategy is to design and use systems that can be temporarily disabled during potentially inappropriate time intervals. However, this approach can be self-defeating. Turning off a mobile phone prevents unimportant interruptions, but it also prevents interruptions that could have conveyed critically important information. Because current systems generally do not have a mechanism for weighing the importance of information against the appropriateness of an interruption, people are forced into extremes of either allowing all interruptions or forbidding all interruptions. This problem is amplified because people forget to re-enable systems after a potentially inappropriate time interval has passed. In one study, Milewski and Smith found that people would regularly set and then forget to clear a busy flag. Their colleagues thus began to ignore the busy flag, because they learned that it was common for people to be shown as busy when they actually were not (Milewski and Smith 2000).

If we could develop relatively robust models of human interruptibility, they might support a variety of significant advances in human-computer interaction and computer-mediated communication. Including a model of human interruptibility in an application need not deprive people of control. For example, a phone could automatically inform a caller that the person being called appears to be busy, then allow the caller to consider their relationship with the person they are calling and the importance of the call to decide whether to interrupt the apparently busy person or leave a message instead (Schmidt et al. 2000). Email and messaging applications might delay potentially disruptive auditory notifications for messages that seem relatively unimportant, but never prevent delivery of the information. Information displays might choose between several methods of conveying information according to the current appropriateness of each method of presentation. Many other specific applications could be designed for different domains. For example, information about interruptibility might be combined with information on expertise and other relevant factors to automatically route incoming technical support requests to the most appropriate member of a technical support staff.

1.2 Research Summary and Components

This dissertation presents work to develop sensor-based statistical models of the interruptibility of office workers in their normal work environments. Our work is based on collecting data in realistic environments and directly measuring and modeling human interruptibility. There are three major components to the research presented in this dissertation:

> First, a novel method is developed, applying a Wizard of Oz technique to inform the top-down development of sensor-based statistical models.
>
> Second, a series of studies are conducted, examining the feasibility and then the robustness of sensor-based statistical models of human interruptibility.

> Third, a software toolkit is created, enabling non-expert development of applications that learn and use sensor-based statistical models of human situations.

Interruptibility is an inherently abstract concept, so this dissertation explores models based on two different approaches to measuring interruptibility. First, we use an experience sampling technique to collect interruptibility self-reports from office workers in their normal work environments. In a series of studies, we show that practical sensors can support learned statistical models that identify situations that office workers report to be "Highly Non-Interruptible" significantly better than human observers. We explore a second measure of interruptibility by developing a controlled study to measure how long people choose to defer a pending interruption. Informed by these studies, we develop `Subtle`, a toolkit for *Sensing User Behavior To Learn about the Environment*. `Subtle` enables the non-expert development of applications that learn and use sensor-based statistical models of human situations by providing an extensible library of appropriate sensors, fully-automated iterative feature generation and selection, continuous learning of personalized models, privacy filtering of collected sensor logs, and support for field deployments of applications. By making research on sensor-based statistical models of human situations accessible to the wider human-computer interaction research community, `Subtle` enables research into how such models can best be included in applications.

1.3 Dissertation Organization

This chapter has motivated our investigation of sensor-based statistical models of human interruptibility and provided a brief overview of the research we will present.

Chapter 2 presents a review of related work, ranging from laboratory studies of interruptibility to deployed systems that model availability. In order to best illustrate how our work is related to that of other researchers, we include our own work in this review.

Chapter 3 introduces the novel Wizard of Oz technique we will use throughout this dissertation. Examining models based on simulated versions of potential sensors enables effective top-down development of sensor-based statistical models of human situations.

Chapter 4 presents a Wizard of Oz analysis of audio and video recordings collected in the normal work environments of four office workers. Using an experience sampling technique to collect occasional self-reports of a person's interruptibility, we simulate the presence of many potential sensors and examine the feasibility of learning statistical models to identify "Highly Non-Interruptible" situations.

Chapter 5 explores the reliability of our learned statistical models by comparing them to the reliability of human observers. Using snippets of our collected audio and video recordings to measure the reliability of human observers, we show that the previous chapter's learned statistical models based on simulated sensors can identify "Highly Non-Interruptible" situations significantly better than human observers.

Chapter 6 extends this result by deploying actual, implemented sensors with a larger and more diverse set of office workers. While different features are most useful for different types of office workers, we show that a single model of our diverse office workers can identify situations reported as "Highly Non-Interruptible" significant better than human observers. We also show that a model built entirely from software-based sensors for a typical laptop computer, including analyses of audio from a laptop's built-in microphone, can support models that identify "Highly Non-Interruptible" situations significantly better than human observers.

Chapter 7 more carefully examines task engagement in sensor-based statistical models of human interruptibility. While the results of our previous studies were largely dominated by results related to social engagement, this chapter controls for social engagement and examines programmers working on a natural programming task. We start by using our Wizard of Oz technique to analyze screen capture recordings, then develop a sensing plug-in to capture low-level events in the programmer's development environment. We then learn models that can identify situations in which a programmer will choose to delay attending to a pending interruption.

Chapter 8 applies the results of our studies to the development of `Subtle`. Based on the sensors and features proven effective in our studies of human interruptibility, `Subtle` provides fully-automated iterative feature generation and selection with an extensible sensing library appropriate for a typical laptop computer. Using `Subtle`, an application developer can include a sensor-based statistical model of a human situation in as little as

16 lines of code. This allows human computer interaction researchers to focus on compelling applications and datasets, rather than the difficulties of collecting appropriate sensor data and learning statistical models.

Chapter 9 concludes this dissertation by summarizing some of the major research contributions from each chapter and discussing several areas for future work.

Much of the work presented in this dissertation has previously been published in other forums. Chapter 4's initial Wizard of Oz feasibility study appears in (Hudson et al. 2003). A more complete analysis, together with Chapter 5's study of human observers, appears in (Fogarty et al. 2005b). Chapter 6's study deploying actual, implemented sensors with a larger and more diverse set of office workers appears in (Fogarty et al. 2004a). Chapter 7's work using a development environment's low-level event stream to model programmer task engagement appears in (Fogarty et al. 2005c). We have updated and clarified many aspects of the presentation of this previously published work. Publications are currently being prepared for Chapter 3's discussion of our Wizard of Oz method and Chapter 8's discussion of `Subtle`.

Chapter 2

Related Work

This chapter, broken into six sections, provides a general overview of the state of research around sensor-based statistical models of human interruptibility. The first section discusses work examining how people currently perceive and manage interruptions. The second section discusses laboratory studies of interruptions in controlled environments. The third section discusses context-aware computing and human activity recognition, including some research that has examined activity recognition with a potential relationship to human interruptibility. The fourth section discusses deployed systems that are related to interruptibility. The fifth section, which includes the studies we present in this dissertation, discusses work to explicitly measure and model human interruptibility using data collected in realistic environments. Finally, the sixth section discusses a variety of methods and tools that are not directly related to human interruptibility but are used in this dissertation. Related work is also discussed throughout the dissertation as appropriate.

2.1 Research on How People Perceive and Manage Interruptions

McFarlane and Latorella review a variety of work motivating the study of interruptions in human-computer interaction (McFarlane and Latorella 2002). They argue that further research on interruptions is motivated by the increasing complexity of human-computer systems and by the use of automated monitoring as a strategy to increase the amount of

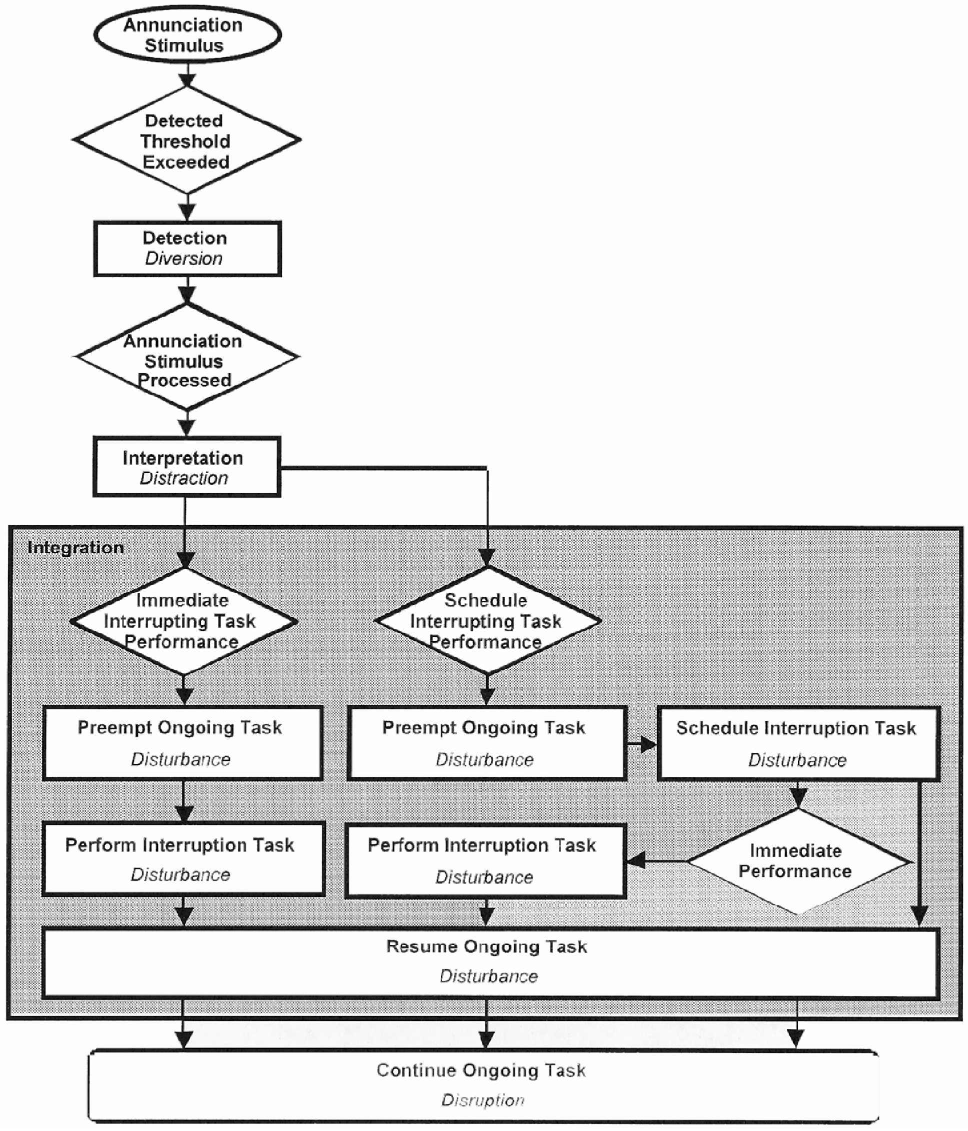


Figure 2.1. Latorella's interruption management stage model (McFarlane and Latorella 2002).

information that a person can manage. McFarlane and Latorella focus on domains where the costs of failure are very high, including interruptions of pilots and interruptions of operators of the Aegis Combat System, but their discussion is intended to be more general. As a basis for understanding and studying interruptions, they review Latorella's Interruption Management Stage Model, shown in Figure 2.1. This model provides an organizing framework for examining how people respond to interruptions. For example, Figure 2.1 shows four effects of interruptions (diversion, distraction, disturbance, and disruption) and where these effects can occur in the process of an interruption.

McFarlane and Latorella also review McFarlane's taxonomy of human interruption, which describes eight major dimensions in the problem of human interruption: the source of interruption, the individual characteristic of the person receiving the interruption, the method of coordination, the meaning of the interruption, the method of expression, the channel of conveyance, the human activity changed by the interruption, and the effect of the interruption. In contrast to McFarlane and Latorella, this dissertation is explicitly focused on the interruptibility of office workers. Although we recognize the importance of the many dimensions of interruptions discussed by McFarlane and Latorella, this dissertation is focused on how sensed context relates to the cost of delivering an interruption.

Perlow studied the time usage of a group of 45 software engineers in a technology corporation (Perlow 1999). She found a crisis mentality that creates a self-perpetuating cycles of interruptions. In this cycle, a person up against a critical deadline feels free to interrupt a colleague and request assistance. But this delays the scheduled work of the interrupted colleague, thus causing the colleague to fall behind in their own work. The colleague is then forced to interrupt yet another person. Perlow also found a lack of management recognition for the people who respond to such requests for assistance, as their managers recognized that they were behind on their work but did not appreciate that this was because they had been assisting somebody else. In an attempt to help address these issues, Perlow introduced scheduled quiet times into the working week of the group. During these quiet times, the engineers were supposed to work alone and without disturbing other members of the group. This did seem to improve group productivity, but the effect was short-lived because there was no incentive for the engineers to abide by quiet time and so the self-perpetuating cycle reemerged.

Hudson *et al.* use an experience sampling method and qualitative interviews to examine how corporate research managers perceive and manage interruptions, finding a tension between the value delivered by appropriate interruptions and the disruption caused by inappropriate interruptions (Hudson et al. 2002). Among their findings, Hudson *et al.* report that managers sometimes move away from their computer or even away from their office in order to obtain uninterrupted working time. But they note that managers reported taking such extreme measures for only part of a day, as their availability to address interruptions is an important requirement of their work.

O’Conaill and Frohlich conducted an observational study of interruptions in the workplace, analyzing video that Whittaker *et al.* previously collected by following two office workers for a week (Whittaker et al. 1994; O'Conaill and Frohlich 1995). Examining 125 naturally occurring interruptions, they found that the initiator of an interruption benefited from 76% of interruptions while the person being interrupted benefited from only 64% of interruptions. O’Conaill and Frohlich also found only 55% of interruptions resulted in the interrupted person resuming the work they were doing prior to the interruption (though they only consider immediate resumption, not whether the interrupted work was later resumed).

Czerwinski *et al.* conducted a diary study examining how 11 participants interleave multiple tasks amidst interruptions (Czerwinski et al. 2004). Among their findings, participants reported using a conservative average of 1.75 documents per activity. They found that 40% of reported task switches were self-initiated, 19% corresponded to addressing an item on a physical or electronic to-do list, 14% were due to a phone call, and 10% were due a meeting or appointment. These results inform the development of GroupBar, a prototype taskbar enhancement that allows users to group and organize documents and windows related to a task.

Mark *et al.* present a observational study of 24 information workers, examining their work fragmentation by considering the length of time spent in an activity and the frequency of interruptions (Mark et al. 2005). This observational approach used in this work examined activities at a much finer granularity than the diaries collected by Czerwinski *et al.* They consider both external and internal interruptions, defined by Miyata and Norman as interruptions that are caused by an event in the environment versus by an internal decision to stop working on a task (Miyata and Norman 1986). Mark *et al.* found that people spend only short periods of time in a working sphere before switching to another, perhaps because they are continually juggling their priorities according to external workplace demands. They also found that 77% of interrupted work was resumed at some point before the end of the same day. Based on these and other findings, Mark *et al.* discuss requirements for supporting fragmented work. These are that interruptions should match the current working sphere, making them of benefit instead of a disruption, that one should be able to easily and seamlessly switch between tasks, and that interrupted tasks should be easily resume by preserving the state of the task when it was interrupted and by providing cues for reorienting to the task.

In the context of this dissertation, the field studies presented in this section help to motivate the study of interruptions. Prior work has shown that interruptions are both disruptive to work and critical to obtaining the information needed to work effectively. If systems could deliver interruptions at more appropriate times, we might be able to obtain the benefits provided by the information delivered in interruptions while minimizing the cost of their delivery.

2.2 Laboratory Studies of Interruptions in Controlled Environments

A number of laboratory studies have examined the effects of interruptions in controlled tasks, sometimes with the goal of understanding human memory or cognitive processes, other times focusing on the development of guidelines for human-computer interaction. These studies often provide valuable high-level insight into the design of applications, but it can sometimes be difficult to apply this insight to a particular problem. In contrast, this dissertation is focused on the development of sensor-based statistical models that can be directly incorporated into applications.

Zeigarnik was the first to publish on the relationship between interruptions and selective memory (Zeigarnik 1927/1938). In what has become known as the Zeigarnik Effect, people can recall the details of interrupted tasks better than tasks that have been completed, potentially due to a tension which persists only until the needs of the interrupted task are satisified. Zeigarnik began to investigate this effect after informally noticing that a waiter could recall large numbers of orders, but only those that had not yet been completed.

Gillie and Broadbent present a series of studies using a computer to administer and interrupt a text-based navigation and object collection task (Gillie and Broadbent 1989). They examine the effect of the length of the interruption, the similarity of the interruption to the main task, and the complexity of the processing demanded by the interruption. Their findings suggest that the length of an interruption is not a major factor in whether the interruption will be disruptive, but that even short interruptions are likely to be disruptive if they are similar to the interrupted task or are complex enough to require significant processing or memory storage.

Hess and Detweiler showed that the effects of an interruption can be strongly influenced by expertise or training in handling the interruption (Hess and Detweiler

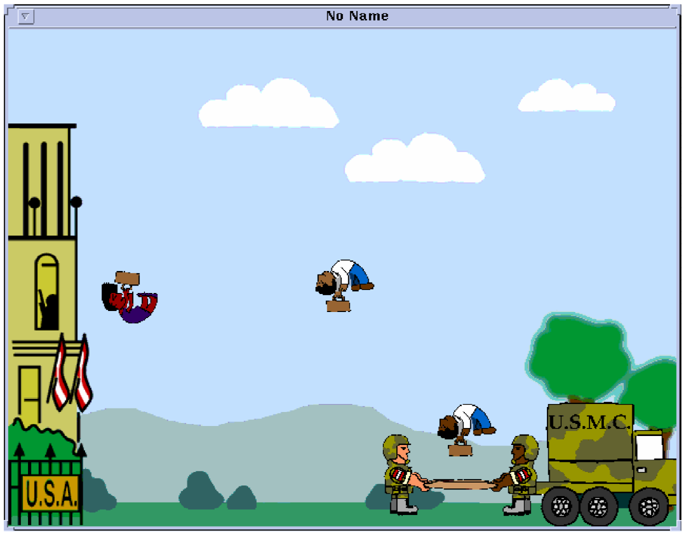


Figure 2.2. McFarlane's primary task, requiring continuous attention to successfully bounce people to safety (McFarlane 1999; McFarlane 2002).

1994). Interruptions similar to a primary computer task were very disruptive in the first two of three sessions, but they were significantly less disruptive in the third session. In order to establish that the effect was not due to practice on the primary task, Hess and Detweiler examined training for two sessions on the primary task without interruptions and then introducing the interruptions in the third session. The introduction of the interruptions was significantly harmful to the task, even though the participants were highly trained on the task. This indicates that expertise or training in handling the interruption is important, not just expertise or training in the primary task.

McFarlane tested four known methods for coordinating the delivery of interruptions: immediate, negotiated, mediated, and scheduled (McFarlane 1999; McFarlane 2002). Working at a computer, participants focused on a primary task modeled after the Nintendo video game *Fire*, which requires the player manage jumpers across the screen by bouncing each jumper three times at three different locations. Interruptions were introduced by a secondary task requiring the participant to solve a matching problem. McFarlane showed that the negotiated coordination of interruptions generally obtained the best result, as long as small differences in the time taken to begin addressing an interruption are not critical. But he also notes that the use of negotiated coordination could result in an interruption being indefinitely delayed.

Robertson *et al.* examine immediate and negotiated interruptions to help users debug errors in formulas during end-user spreadsheet programming (Robertson et al. 2004). They show that participants in the negotiated condition learned more about formula propagation, were able to fix more bugs per minute, and had a better understanding of whether they had fixed bugs. These differences were not due to a simple difference in the time required to attend to immediate interruptions, but rather seem to be because the different interruption styles led participants to pursue different debugging strategies.

Dabbish and Kraut examine interruption coordination in teams using a task similar to McFarlane's (Dabbish and Kraut 2004). A helper partner focuses on McFarlane's task while responding to occasional requests for information from the asking partner. The asking partner has either a detailed display of the helper partner's task, an abstract display of the helper partner's workload, or no awareness of the helper partner's workload. Dabbish and Kraut found that the abstract awareness display provided much of the benefit of a detailed display while imposing less attentional cost on the interrupter. They also found that the presence of a team reward structure was required before the asker would consider the helper's interruptibility.

In a series of studies, Czerwinksi, Cutrell, and Horvitz examine the effect of interruptions due to notifications delivered during list searching tasks and simple office productivity tasks (Czerwinski et al. 2000a; Czerwinski et al. 2000b; Cutrell et al. 2001). They showed a harmful effect on overall task performance for notifications delivered while a participant is typing, using buttons or menus, or evaluating search results. While they found that interruptions delivered early in a task had less negative impact on the overall time required to a complete a task, they also found that interruptions delivered early in a task resulted in a more negative impact on participant memory of the interrupted task.

Adamczyk and Bailey study interruptions at different points in a task hierarchy, examining the relationship between predicted cognitive load and how a participant perceives an interruption (Adamczyk and Bailey 2004). They start by eliciting hierarchical models of how participants describe three office tasks: a document editing task, a video summarization task, and a web search task. Different participants then performed the tasks while being interrupted at different points in the task hierarchy, with the expectation that the best time for an interruption will be between subtasks that are

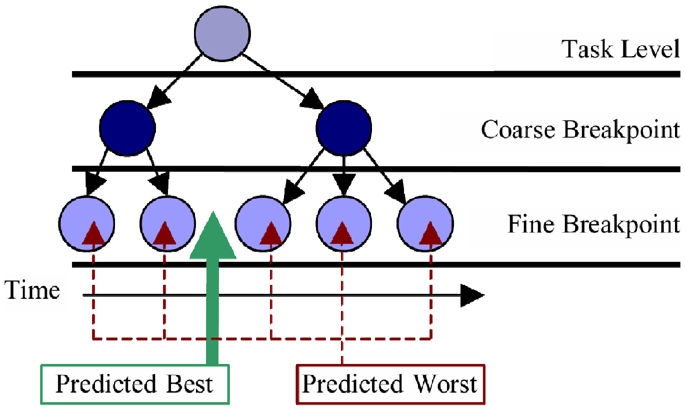


Figure 2.3. Adamczyk and Bailey found that participants are more receptive to interruptions between subtasks high in a task hierarchy and that more damage is caused by interruptions that occur during subtasks that are deep in the hierarchy (Adamczyk and Bailey 2004).

high in the task hierarchy, while the worst time to interrupt will be during subtasks that are deep in the task hierarchy. While they did not find differences in task time, subjective questionnaires indicate that interruptions at the predicted best times produce less annoyance, frustration, and time pressure, require less mental effort, and are deemed more respectful of the primary task. Further, interruptions at the predicted worse times result in responses that are worse than responses for random interruptions.

Iqbal *et al.* examine hierarchical task models using pupil dilation to measure mental workload (Iqbal et al. 2005). Studying a route planning task and a document editing task, they developed and validated GOMS models of the tasks. Because participants took different lengths of time to complete the tasks, Iqbal *et al.* manually aligned their recordings with the GOMS models, using screen recordings together with keyboard, mouse, and gaze activity to track each participant's progress through the task model. They found that different types of subtasks imposed different workloads. For example, in the route planning task, Reasoning subtasks induced more workload than Store or Recall subtasks. They also showed that workload decreases at subtask boundaries, and that it decreases more at boundaries higher in the task hierarchy (consistent with the results of Adamczyk and Bailey). But they also found that mental workload is not predicted simply by the subtask type or its level in the hierarchy, as tasks at the same level in the hierarchy can have different workloads. Using this type of an approach to manage interruptions is therefore likely to require measurement of workload, not just the specification of a task model.

In the context of this dissertation, the controlled studies of interruptibility presented in this section serve two roles. One on hand, these studies show that significant improvements can be obtained by carefully accounting for interruptibility in controlled situations. This motivates our work to realize these benefits in realistic field environments. On the other hand, the results of these controlled studies are only directly applicable in the constrained and sometimes artificial situations in which the studies are conducted. Our work aims to develop models of interruptibility that are practical for real-world deployment. For example, this dissertation does not assume the availability of detailed task models. Instead, we focus on modeling interruptibility in a random sampling of situations experienced by typical office workers. It is therefore more apparent how our results directly apply to deploying interruptibility models in office environments.

2.3 Context-Aware Computing and Human Activity Recognition

While laboratory studies of interruptions can provide valuable insight into cognitive processes or the design of applications, they cannot by themselves enable the development of applications that adapt to human context. A variety of work has examined the recognition of human activities, and reviewing all of it is beyond the scope of this section. This section discusses some of the most relevant work on human activity recognition, while later sections will discuss work that focuses on directly measuring and modeling human interruptibility.

Horvitz *et al.* present Lumière, a system that reasons about the goals and needs of users in order to provide intelligent assistance within an application (Horvitz et al. 1998). Working from a model of likely goals within an application, Lumière monitors an application's event stream to infer what goals a person may be pursuing and whether the person is likely to benefit from assistance. A personal competency profile, updated when when a person views documentation or successfully completes tasks, is also considered in deciding whether a person is likely to benefit from assistance. The likelihood that a user needs assistance is compared to a user-adjustable threshold to determine whether the system should attempt to provide assistance. Lumière's knowledge of a person's current goals can also be used to inform the results of user-initiated help requests, as likely goals can be considered together with the text of a query.

Schmidt introduces implicit interaction, discussing how a variety of contextual information can trigger implicit changes in an interaction (Schmidt 2000). Dey *et al.* define context as "any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves" and develop the Context Toolkit to support the investigation of research challenges in the area of context-aware computing (Dey et al. 2001).

Location is widely considered an important piece of context that has prompted significant research. The PlaceLab initiative is investigating privacy-appropriate approaches to estimating a device's location using existing radio beacons, such as WiFi access points and mobile phone towers (Schilit et al. 2003; LaMarca et al. 2005). Patterson *et al.* infer transportation-related activities, such as bus riding, from a stream of GPS data (Patterson et al. 2003). Ashbrook and Starner use GPS data to infer significant locations and patterns of movement across users (Ashbrook and Starner 2003). Liao *et al.* recognize that a particular person's location can be a strong indicator of that person's activity, using GPS data to recognize such activities as being at home, working, shopping, and dining out (Liao et al. 2005).

Several advanced laboratories have been developed to recreate home environments appropriate for studying human activity recognition. The Aware Home, at Georgia Tech, includes ubiquitous computing technology to support a variety of research based on human activity recognition (Abowd and Mynatt 2000). Informed by work on installing large numbers of simple sensors into existing homes (Munguia Tapia et al. 2004), the PlaceLab project, at MIT, is a living laboratory that enables data collection in support of researchers developing context-aware and ubiquitous computing technologies (Intille et al. 2005). Wilson and Atkeson instrument a home to examine work on simultaneously tracking and recognizing the activities of occupants using simple, anonymous sensors like motion detectors and contact switches (Wilson and Atkeson 2005).

Oliver *et al.* develop SEER, which used layered Hidden Markov Models to recognize a set of activities related to human interruptibility (Oliver et al. 2002). Considering audio and video analyses together with the desktop event stream, SEER classifies an office situation into one of six categories: phone conversation, presentation, face-to-face conversation, user present and engaged in some other activity, distant conversation

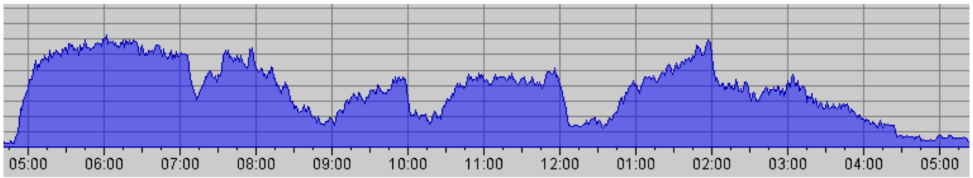


Figure 2.4. A percent-active plot, showing the likelihood a person is active on their computer by time of day (Begole et al. 2002; Begole et al. 2003).

(outside the view of the camera), or nobody present. While work discussed in the next section shows that recognizing these activities is useful in a model of human interruptibility, Oliver *et al.* do not collect a measure of interruptibility or examine how their recognized activities relate to interruptibility. Oliver and Horvitz explore selective perception within Selective SEER, which enables or disables sensing subsystems based on the expected value of the information delivered by that subsystem (Oliver and Horvitz 2003). Selective SEER is able to reason about whether the value provided by a sensing subsystem justifies the computational cost of enabling the system. Subsystems with a low computational cost might therefore always be enabled, while subsystems with a relatively high computational cost (such as vision-based subsystems) might be enabled only when they are likely to provide important information.

Begole *et al.* use minute-by-minute records of computer activity to visualize and model patterns in a person's presence at their computer (Begole et al. 2002; Begole et al. 2003). Figure 2.4 presents an example data visualization, showing that there are clearly times when a particular person is more or less likely to be active on their computer. This type of visualization could be directly incorporated in some applications, but other applications may require the automated recognition of major trends. Using an Expectation Maximization clustering algorithm (Dempster et al. 1977), Begole *et al.* cluster periods of inactivity to model significant trends. They are able to detect such patterns as recurring meetings and the length of a commute (the time between when a telecommuter logs off a home computer until they log into a computer at work).

Avrahami and Hudson model responsiveness to instant messages: whether or not a person is likely to respond to an attempt to initiate communication via an instant message (Avrahami and Hudson 2006). Based on a history of a person's computer activity and whether or not they have previously responded to instant messages, Avrahami and Hudson develop statistical models to predict whether a person is likely to respond to an

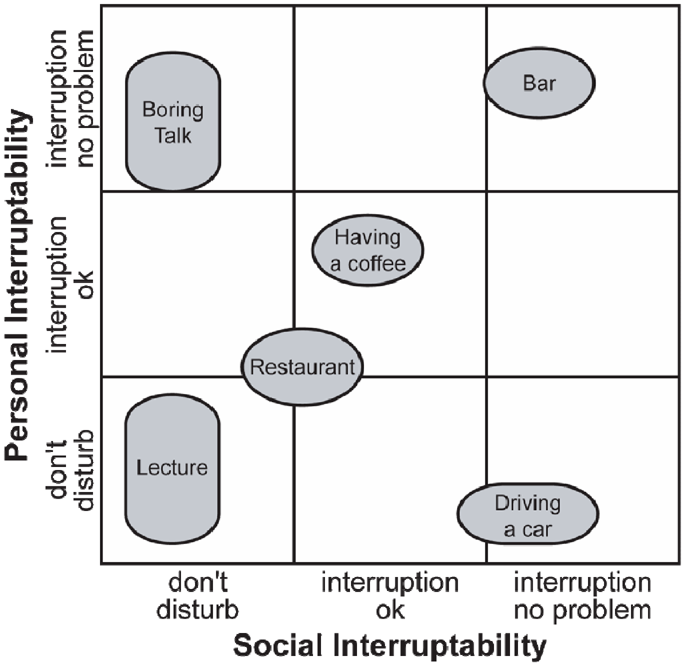


Figure 2.5. Kern *et al.* distinguish between personal and social interruptibility (Kern and Schiele 2003; Kern et al. 2004).

instant message from a person with whom they are not already in a conversation. Their model is neither a model of presence (as with Begole *et al.*) nor a model of interruptibility (as examined in this dissertation), as both of these factors contribute to whether or not a person will respond. Instead, Avrahami and Hudson describe the modeling of responsiveness as an example of modeling demonstrated availability.

Kern *et al.* propose the use of wearable sensors, including a microphone and accelerometers, to model what they refer to as *personal* interruptibility and *social* interruptibility (Kern and Schiele 2003; Kern et al. 2004). They define personal interruptibility as "the interruptibility of the user" and social interruptibility as "the interruptibility of the user's environment," providing the example that an audio notification might be acceptable to a person who is currently bored but that same notification might be very socially disruptive to the meeting they are attending. Kern *et al.* directly model these concepts based on wearable sensors, but we include this work in our discussion of activity recognition because they have conducted only limited analyses of field data collected with their wearable system. Specifically, their initial work is based on an assumed and untested relationship between specific contexts and interruptibility, such as assuming that social interruptibility is high when walking on the street (Kern and

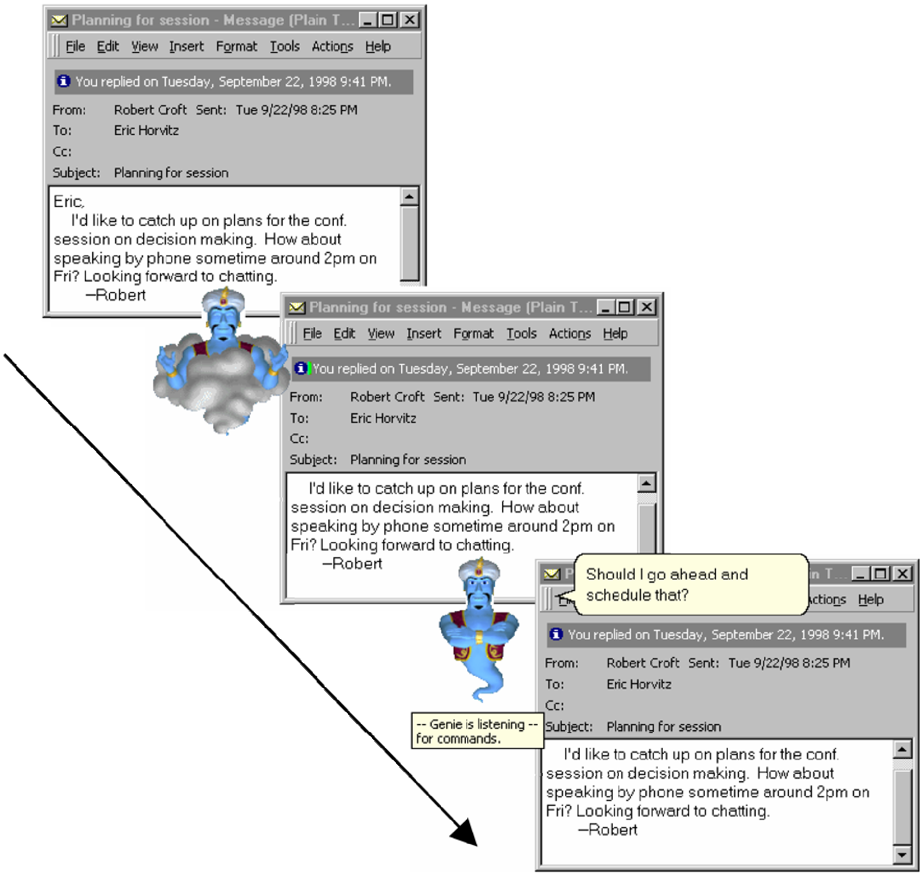
Schiele 2003). Their later work asks participants to estimate the personal and social interruptibility of different contexts, but is focused on participant evaluation of interruptibility only in hypothetical situations (Kern et al. 2004). In contrast, the work presented in the remainder of this chapter directly models interruptibility as measured in field environments or during the execution of natural tasks.

In the context of this dissertation, the context recognition work presented in this section demonstrates a variety of approaches to context-aware computing. We note that typical context-aware systems either define direct mappings between contexts and how a system should behave or directly model a single concrete concept, while this dissertation will focus on the more abstract notion of modeling interruptibility. Part of the attraction of this more abstract concept is the potential for generally useful interruptibility models that can be applied in a variety of applications. This both means that it may not be necessary to develop many different models and that programmers may not need to directly manage many different pieces of sensed context. Instead, programs can be written against a learned model of a person's interruptibility.

2.4 Systems Related to Human Interruptibility

Horvitz presents the LookOut system in the context of discussing principles for the design of mixed-initiative user interfaces (Horvitz 1999). While prior work generally considered intelligent automation and direct manipulation to be different approaches to human-computer interaction, Horvitz discusses mixed-initiative interfaces as an elegant coupling of the two. Among other requirements, Horvitz argues that successful mixed-initiative interfaces will consider uncertainty about a user's goals, consider user attention, employ dialog to resolve uncertainty, allow efficient direct invocation or termination of intelligent automation, and design interactions that minimize the cost of poor decisions by an intelligent system. He discusses these requirements in the context of the LookOut system, providing such examples as automated recognition and support for email messages related to calendar scheduling.

Horvitz *et al.* introduce Priorities in their discussion of attention-sensitive alerting systems (Horvitz et al. 1999). This work examines automated reasoning about the tradeoff between the value of the information gained by delivering a message versus the cost of interrupting to deliver it. While the notion of this tradeoff is more general,


Planning for session - Message (Plain T...
File Edit View Insert Format Tools Actions Help
Planning for session - Message (Plain T...
File Edit View Insert Format Tools Actions Help
Actions Help

You replied on Tuesday, September 22, 1998 9:41 PM.

From: Robert Croft Sent: Tue 9/22/98 8:25 PM
To: Eric Horvitz
Cc:
Subject: Planning for session

Eric,
I'd like to catch up on plans for the conf. session on decision making. How about speaking by phone sometime around 2pm on Fri? Looking forward to chatting.
—Robert

You replied on Tuesday, September 22, 1998 9:41 PM.

From: Robert Croft Sent: Tue 9/22/98 8:25 PM
To: Eric Horvitz
Cc:
Subject: Planning for session

I'd like to catch up on plans for the conf. session on decision making. How about speaking by phone sometime around 2pm on Fri? Looking forward to chatting.
—Robert

Should I go ahead and schedule that?

-- Genie is listening -- for commands.

You replied on Tuesday, September 22, 1998 9:41 PM.

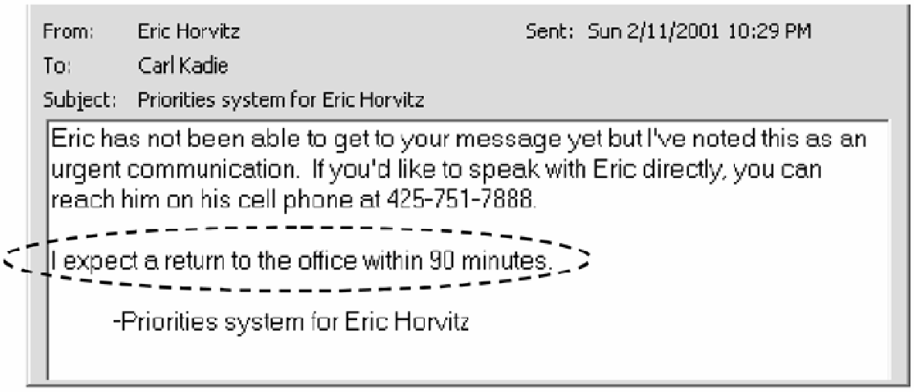
From: Robert Croft Sent: Tue 9/22/98 8:25 PM
To: Eric Horvitz
Cc:
Subject: Planning for session

I'd like to catch up on plans for the conf. session on decision making. How about speaking by phone sometime around 2pm on Fri? Looking forward to chatting.
—Robert

Figure 2.6. Horvitz discusses mixed-initiative user interfaces as the effective coupling of direct manipulation and intelligent automation (Horvitz 1999).

Horvitz *et al.* focus this work on assigning cost according to a sensor-based statistical model of attentional status and estimating value through text analyses of incoming email. They develop classifiers to estimate the criticality of incoming email messages and suggest a variety of applications based on reasoning about the tradeoff between this criticality and different paths of action. For example, Priorities can forward a critical message to a mobile device, play different audio notifications for messages of different criticality, and decide whether to automatically open and display a message. Priorities also enables a smart out-of-office application, wherein the sender of a critical email can be automatically notified that the recipient is currently unavailable and given an estimate of when the recipient is likely to be available to read the message (Horvitz et al. 2002).

Horvitz *et al.* build upon the approach introduced in Priorities while developing the Notification Platform, a multi-device system with an SDK for adding information sources



From: Eric Horvitz Sent: Sun 2/11/2001 10:29 PM
To: Carl Kadie
Subject: Priorities system for Eric Horvitz

Eric has not been able to get to your message yet but I've noted this as an urgent communication. If you'd like to speak with Eric directly, you can reach him on his cell phone at 425-751-7888.

I expect a return to the office within 90 minutes.

-Priorities system for Eric Horvitz

Figure 2.7. An smart "out-of-office" response automatically generated based on Priorities's availability forecasting (Horvitz et al. 2002).

and devices (Horvitz et al. 2003b). An important aspect of the Notification Platform is Coordinate, a core component for reasoning about the cost of interruption and the time until a person is likely to be active on a device (Horvitz et al. 2002). Coordinate considers computer activity, electronic calendar information, audio and video analyses, and location information to reason about a person's availability on multiple computing devices. Using activity logs collected from multiple devices, Coordinate can reason about how long a person is likely to remain available on a device or how long it may be before a person becomes available on a device. Coordinate also conducts several analyses of electronic calendar information, including explicit models of the likelihood that a person will attend an event on their calendar and the cost of an interruption while attending the event. These supervised calendar analyses are based on using forms to collect user indications of whether they are likely to attend meetings and the likely cost of an interruption during a meeting.

Further work by Horvitz *et al.* has examined the deployment of systems informed by this work, including BestCom (Horvitz et al. 2003a; Gibbs 2005) and Bayesphone (Horvitz et al. 2005). Built on Microsoft's Enhanced Telephony prototype (Cadiz et al. 2004), BestCom monitors incoming phone calls and intelligently routes them after considering a variety of criteria. For example, an incoming call from a family member or supervisor might be immediately routed to a person's phone. The response to other callers can then be based on a person's availability, giving callers an indication of when a person is likely to be available or asking them to leave a message. Bayesphone enables the integration of computationally-limited devices like mobile phones into systems like BestCom. Bayesphone runs on a desktop computer, analyzing and pre-compiling policies for how a person's mobile phone should behave during scheduled items on a person's electronic calendar. These pre-compiled policies are downloaded to the phone as a part of regular data synchronization, allowing the computationally-limited device to intelligently handle incoming calls. Bayesphone can also compile policies indicating situations in which it will be useful to collect training data from a user, such as whether or not a person is attending a particular event on their calendar. When the phone is next synchronized with the full system, this targeted training data can be integrated into the system's model of a person's availability.

Mynatt and Tullio discuss Ambush, a calendar system extension that uses a Bayesian model to predict the likelihood of a person's attendance at an event on their schedule (Mynatt and Tullio 2001). While these estimates can be used as input to a variety of higher-level inference problems (as in work by Horvitz *et al.*), they can also be directly presented to users. Mynatt and Tullio explore a variety of methods for visualizing confidence estimates in an electronic calendar, including manipulating the transparency of a calendar entry and using feature maps to illustrate the features dominating a prediction. Tullio *et al.* integrate this type of forecasting into Augur, a shared electronic calendar system, and investigate how people use and react to the addition of intelligent parsing and attendance forecasting (Tullio et al. 2002).

Shell *et al.* discuss attentive user interfaces, including the use of eye gaze to indicate whether a person is currently attending to a computing device (Shell et al. 2003). They use an IBM PupilCam to detect eye gaze, which uses computer vision to detect pupils and determine whether people are looking at the sensor (Morimoto et al. 2000). They demonstrate such applications as looking at a light and speaking "on" or "off." While these phrases would normally be ambiguous in a room containing more than one light, eye gaze can be used to indicate what device a person wants to interact with.

McCrickard *et al.* examine the evaluation of notification systems, which deliver information of interest in a parallel, multitasking approach that is extraneous or supplemental to a user's attention priority (McCrickard et al. 2003). They develop a three-dimensional framework for comparing and categorizing notification systems, based on interruption, reaction, and comprehension. This framework helps to focus notification system work on understanding user goals surrounding the degree to which a primary task is interrupted by a notification system, whether a notification system requires rapid and

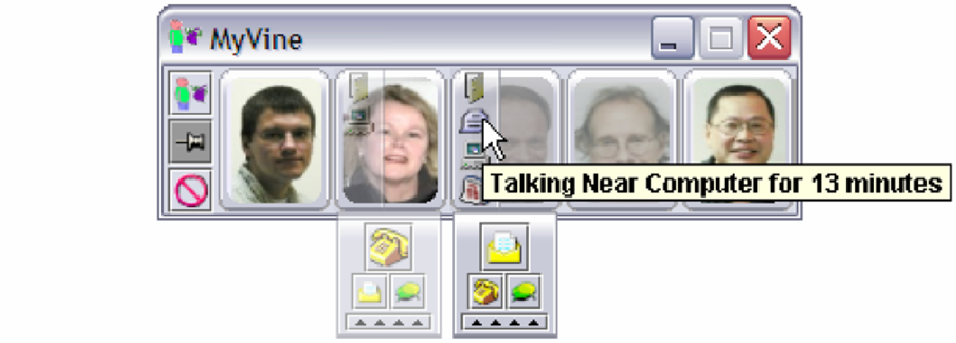


Figure 2.8. Fogarty *et al.* develop and study MyVine, a context-aware communication client with a model of availability (Fogarty et al. 2004b).

accurate perceptual reaction to the onset of a notification, and whether notifications lead to long-term comprehension of presented information.

Fogarty *et al.* develop and study MyVine, a context-aware communication client that uses a person's location, electronic calendar, detection of nearby conversations, and level of computer activity to estimate availability for communication (Fogarty et al. 2004b). This work raises an interesting tension over how people will interpret such context when attempting to initiate communication, as an indication that a person is not interruptible does not necessarily mean that colleagues will choose not to interrupt. After all, many existing systems only provide indications of presence and so people are accustomed to using context only to decide whether a person is present. Further, people who clearly have a need to interrupt may not consider their particular interruption to be disruptive, consistent with Perlow's finding that engineers in her study felt they had no incentive to abide by a quiet time policy (Perlow 1999).

Begole *et al.* develop Lilsys, which integrates work on sensor-based models of availability into a previously-developed context-aware instant messaging client (Tang et al. 2001; Begole et al. 2004). They provide a set of passive sensors for inferring availability, but also provide such options as a manual override, allowing people to specifically indicate that they are not available for a length of time. Consistent with prior work indicating that people do not use or forget to set such manual indications of availability (Milewski and Smith 2000), Begole *et al.* report that participants did not use this manual override functionality. They also found that an indication a person was not interruptible did not prevent colleagues from sending instant messages, but note that colleagues may have instead shaped their interruption differently. For example, a colleague might send a message of the form "Can you call me when you're free?"

Bailey *et al.* develop a task-monitoring framework based on PETDL, a Pattern-based Event and Task Description Language (Bailey et al. 2005). They assume the availability of high-level application events like those in plug-in architectures for some applications and create tools for specifying tasks in terms of these events. At runtime, their framework monitors events and maintains a list of tasks that may be in progress. One of the more interesting characteristics of this framework is an explicit recognition of multitasking. Rather than assuming that all of the events in a task will arrive in a single sequence, Bailey *et al.* account for each incoming event by either assigning it to a task already in progress, by starting a new task to account for it, or by ignoring it.

Avrahami and Hudson develop the QnA instant messaging extension, which monitors incoming and outgoing instant messages to identify and provide salient notifications for the incoming messages that are most likely to require attention (Avrahami and Hudson 2004). QnA searches for the appearance of questions in instant messages and keeps track of whether a user is expecting a response to a question they previously asked. It alerts a user when they receive a non-trivial question in an instant message or when a message is likely to be a response to a previous question. While studies of QnA usage have not yet been presented, informal results suggest that QnA addresses interruptions by allowing people to ignore unimportant messages that arrive while they are busy. A busy person can respond to questions identified by QnA, deferring other messages until a later time.

In the context of this dissertation, the interruptibility-related systems presented in this section motivate our systematic examination of interruptibility models in field environments. Given the difficulties inherent to developing and deploying systems, the work presented in this section is often focused on technical hurdles or qualitative reports of the results of deploying these systems. This dissertation focuses on systematic and quantitative approaches to understanding the reliability of sensor-based statistical models of human interruptibility.

2.5 Studies of Sensor-Based Models of Human Interruptibility

Within the context of prior work on context-aware computing and systems that model concepts related to interruptibility, this dissertation and this section are explicitly focused on studies to examine the reliability of sensor-based models of human interruptibility.

Our Wizard of Oz study of office worker interruptibility, presented in Chapter 4, examines how a variety of potential sensors are related to interruptibility self-reports (Hudson et al. 2003; Fogarty et al. 2005b). We measure interruptibility using an experience sampling technique to collect occasional self-reports on a five-point scale ranging from "Highly Non-Interruptible" to "Highly Interruptible." Learning statistical models from the simulated output of potential sensors, we show that models of human interruptibility can identify "Highly Non-Interruptible" situations significantly better than human observers. Among our findings is the importance of a sensor to detect whether anybody in an office is currently talking, which is a strong indicator that an office worker is not interruptible.

Horvitz and Apacible use a retrospective labeling approach with Dynamic Bayesian Networks to model interruptibility in the Interruption Workbench, which considers audio and video analyses, the desktop event stream, and analyses of a person's electronic calendar (Horvitz and Apacible 2003). After collecting five hours of audio and video recordings from each of three participants, Horvitz and Apacible ask each participant to review the recordings and label spans of time as having a *High*, *Medium*, or *Low* cost of interruption. This research shares our motivation of directly modeling human interruptibility, but differences in the data (our collection of occasional self-reports using experience sampling over several weeks versus Horvitz and Apacible's more fine-grained analysis of a shorter period of time) make it inappropriate to attempt to compare model performance. Among other findings, this work demonstrates the utility of electronic calendars in sensor-based statistical models of human interruptibility. Even though their models include analyses of audio and video input streams (which presumably capture actual social engagement), the planned social engagement captured by electronic calendars provided additional value in models of human interruptibility.

Our robustness study, presented in Chapter 6, deploys actual, implemented sensors to examine models of the interruptibility of a set of office workers with more diverse responsibilities and working environments (Fogarty et al. 2004a). Using the same experience sampling technique as in our prior Wizard of Oz work, we show that a single model of ten office workers with diverse responsibilities and working environments can identify "Highly Non-Interruptible" situations significantly better than human observers. We also show that different features are most useful for different types of office workers and that a typical laptop computer can support robust models.

Horvitz *et al.* discuss BusyBody, which uses an experience sampling technique to collect interruptibility self-reports to build personalized models of interruptibility (Horvitz et al. 2004). Intended for data collection to support a general model of interruptibility that could then be used in other applications, BusyBody provides such functionality as allowing a person to configure the frequency of prompts for self-reports and allowing a person to disable prompts for a length of time. It then uses dynamic Bayesian networks to analyze the relationship between the collected self-reports and sensors related to the desktop event stream, time of day, day of week, electronic calendar events, a microphone-based conversation detection system, and WiFi-based location estimates. In monitoring the desktop event stream, BusyBody considers such high-level temporal concepts as the duration of dwells on windows and applications, the frequency of focus switches between windows and applications during different time intervals, and the overall level of activity in different time intervals. Horvitz *et al.* collect data from four office workers and build models of interruptibility that have reliabilities that seem to be comparable to those developed in our robustness study.

Nagel *et al.* use an experience sampling technique to study interruptibility in the home, with a focus on modeling availability for communication with close friends and family (Nagel et al. 2004). They investigate co-presence, location, and activity as observable factors influencing availability, finding significant effects based on whether a person is alone, in or around the kitchen, engaging in face-to-face conversation, watching television or movies, playing games, managing personal/family information, or engaging is miscellaneous leisure activities. These results suggest both the need to understand the perceived source of an interruption in experience sampling studies and the possibility that models of interruptibility might draw sharp distinctions between different locations.

Our study of task engagement, presented in Chapter 7, examines the use of low-level event streams in a programmer's development environment to automatically extract features indicative of task engagement (Fogarty et al. 2005c). We measure

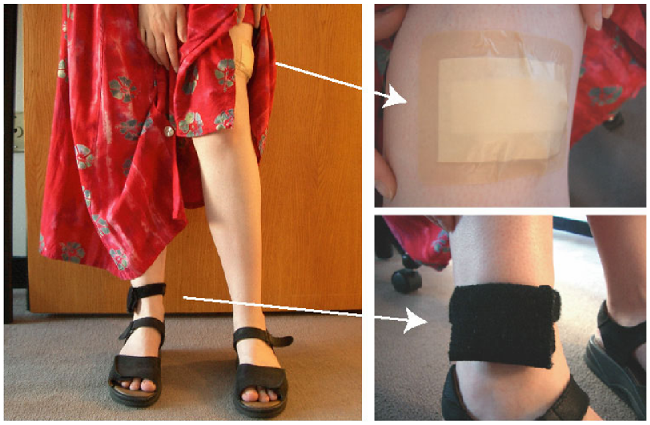


Figure 2.9. Ho and Intille use wireless accelerometers to detect physical activity transitions and measure their relationship to interruptibility (Bao and Intille 2004; Ho and Intille 2005).

interruptibility by using a negotiated coordination of interruptions and timing how long a programmer chooses to defer a pending interruption. The resulting models are an important contribution because prior work has generally been dominated by results related to social engagement or has considered task engagement only when high-level application events are available, while this work shows that task engagement can be captured using automatically generated features based on low-level event streams.

Ho and Intille use a wearable accelerometer-based activity recognition system to examine whether people are more interruptible at physical activity transitions, studying 25 participants who each carried a fully-functional sensor-enabled device throughout a single workday (Bao and Intille 2004; Ho and Intille 2005). They use an experience sampling technique, requesting self reports both at random times and at times when the activity recognition system detected a transition from sitting to walking, walking to sitting, sitting to standing, or standing to sitting. They found that participants reported being significantly more receptive to interruptions at physical activity transitions, though the overall magnitude of the effect was somewhat small. They also describe qualitative results indicating situations where this was definitely not true, including the example of a doctor standing up to walk over to consult with a patient, then being interrupted. Integrating this type of physical activity transition detection with other sensors should prove practically significant and provide a model with the information needed to identify situations where physical activity transitions alone are not a good indicator.

2.6 Methods and Tools Related to this Dissertation

This section discusses some methods and tools that are not necessarily related to human interruptibility but are used in this dissertation. We provide this information and these references here so that the remainder of this dissertation can better focus on our work on human interruptibility.

Wizard of Oz techniques have a long history in speech recognition and the development of intelligent agents (Fraser and Gilbert 1991; Dahlbäck et al. 1993; Maulsby et al. 1993). The techniques take their name from the classic movie in which an impressive system is based entirely on a man behind a curtain. In a traditional Wizard of Oz study, an application designer simulates the intelligence necessary to present an interface to a participant. A designer might read from a script, or a prototyping tool like SUEDE could be used to play pre-recorded responses (Klemmer et al. 2000). This allows an application developer to iterate on a design or examine other qualities of an interface without requiring the implementation of the speech recognition or other intelligent aspects of a system. This dissertation applies a Wizard of Oz technique to simulating the presence of sensors, but studies actual learned statistical models based on the simulated output of these potential sensors.

The experience sampling method, sometimes referred to as ESM or a beeper study, is a method commonly used to study human behavior in natural environments, or *in situ* (Larson and Csikszentmihalyi 1983; Feldman-Barrett and Barrett 2001). At different time intervals, a person is asked to respond to a questionnaire, with modern studies generally using handheld devices or desktop software-based prompts. Experience sampling is an attractive method because it avoids the recall biases and other validity issues that can arise with diary studies or other retrospective approaches. Prompts are generally delivered at random time intervals, as this removes any bias that might be introduced by scheduling the delivery of prompts. But it can be difficult to use completely random prompts to collect data about specific situations, so context-aware experience sampling is sometimes of interest (Ho and Intille 2005).

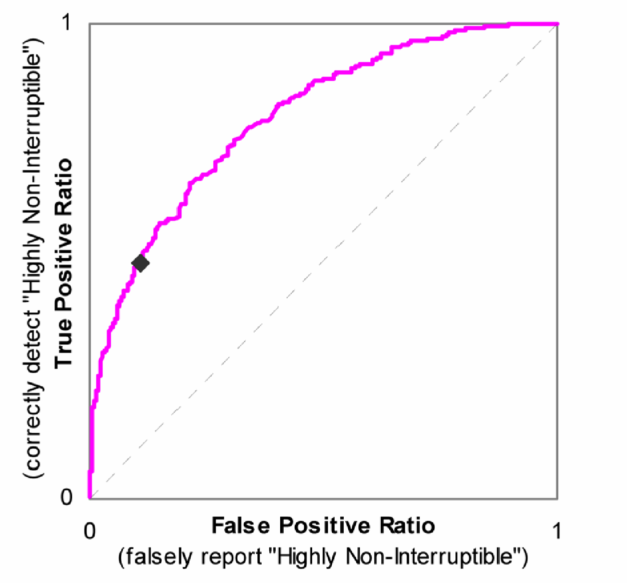


Figure 2.10. An ROC curve illustrates the tradeoff between true and false positives at every potential threshold. In this example, the curve is marked at a threshold for $p > 0.5$, a common threshold for two-choice problems. A curve entirely above the one shown would illustrate a model that is clearly better, as it would generate more true positives for the same number of false positives at every threshold. The area under the curve, A', is a widely used measure of the quality of a model.

Receiver Operating Characteristic (ROC) curve analysis is a technique with a long history in signal detection (Green and Swets 1966) and medical diagnostics (Metz 1978; Hanley and McNeil 1982) that has more recently drawn attention from the machine learning and human-computer interaction research communities (Bradley 1997; Hand and Till 2001; Fogarty et al. 2005a). An ROC curve analysis accounts for the fact that a probability estimate or confidence indicator is typically compared to a threshold before deciding what action to take. Using a lower threshold will reduce false negatives at the cost of increasing false positives. Conversely, using higher threshold will reduce false positives and but increase false negatives. While the relative cost of a false positive or a false negative will vary across applications, an ROC curve plots the tradeoff at every possible threshold. The area under this curve can then be used as a general indicator of the quality of a model. Throughout this dissertation, we will use A', the area under an ROC curve, to evaluate the reliability of sensor-based statistical models. Though this dissertation will provide model accuracies where appropriate, we refer readers interested in our statistical tests to the references provided in this paragraph.

Chapter 3

A Wizard of Oz Approach To Top-Down Sensor Development

3.1 Introduction

Traditional bottom-up sensor development, wherein sensors are developed, deployed, and evaluated, can be expensive and error-prone when applied to creating sensor-based statistical models of human situations. The difficulty lies in the fact that a sensor must meet two distinct requirements: it must reliably detect some piece of context *and* that context must have a meaningful relationship to the concept being modeled. In the case of human interruptibility, for example, an unreliable camera-based object tracker might introduce noise that obscures a meaningful relationship between an object and a person's interruptibility. Conversely, a perfect camera-based object tracker is useless to statistical model of human interruptibility if the tracked objects have no relationship to a person's interruptibility. Traditional bottom-up sensor development focuses on the first of these requirements, but fails to account for the second. If a sensor has little or no relationship to the concept being modeled, this will not be discovered until after the sensor has been developed, deployed, and evaluated. By that point, significant time and resources have likely been wasted on the sensor's development and deployment.

This chapter discusses our novel top-down approach to developing sensor-based statistical models of human situations. Our approach is based in using a Wizard of Oz technique to simulate potential sensors from detailed recordings collected in realistic

- Collect detailed recordings in realistic environments.
- Simultaneously collect a measure of the concept that will be modeled.
- Examine the recordings to develop ideas for sensors that might be predictive of the collected measure.
- Systematically simulate potential sensors from the collected recordings.
- Learn statistical models from the simulated sensors and select potential sensors based on the utility of their simulated versions and their expected cost of implementation.
- Implement the selected sensors.
- Validate the implemented sensors in the deployment environment.

Figure 3.1. The seven steps in our approach the
top-down development of sensor-based statistical models.

environments. We then construct statistical models from the simulated sensors, allowing an early and low-cost measurement of the utility of a potential sensor for modeling the desired concept. A researcher can then weigh the expected utility of a sensor against the time and resources that are likely to be required to develop and deploy the sensor. Our approach is top-down because it focuses on determining what sensors are most appropriate for a particular application, as opposed to building sensors and then testing whether they are appropriate for the application.

The remainder of this chapter summarizes our approach in seven steps. This chapter is intentionally organized as a high-level discussion, as we only intend to provide an overview of the process that we will apply in later chapters. After introducing our approach, this chapter discusses three explicit benefits. First, our approach exposes both dimensions of a tradeoff between sensor complexity and the utility of a sensor in an application. Second, our approach reduces the costs of developing sensor-based statistical models by enabling iterative exploration of potential sensors. Third, our approach allows a separate inspection of errors caused by a shortcoming in a sensor implementation versus errors caused by a shortcoming in a statistical model.

3.2 Our Approach

Our approach to the top-down development of sensor-based statistical models can be summarized in seven steps, shown in Figure 3.1. These steps are:

Collect detailed recordings in realistic environments. In the most straightforward case, our approach can be based on the collection of audio and video recordings from cameras placed in the deployment environment. We will also discuss the use of videos collected with computer screen capture software. In separate work, we have applied our approach using logs containing low-level audio features from microphone-based sensors (Fogarty et al. 2006). While our approach can be based on a variety of types of recordings, the use of a Wizard of Oz technique requires that people can reliably interpret the recordings. Recordings that are difficult to interpret, such as extremely low-level event logs, are likely inappropriate for use with our approach.

Simultaneously collect a measure of the concept that will be modeled. Because we are interested in the supervised learning of sensor-based statistical models, we pair the collection of detailed recordings with a simultaneous collection of a measure of the concept we will be modeling. This dissertation discusses two approaches to collecting measures of interruptibility. We first use experience sampling to collect occasional interruptibility self-reports. We then examine the introduction of controlled interruptions, measuring how long people choose to defer a pending interruption.

Examine the recordings to develop ideas for sensors that might be predictive of the collected measure. While intuition and existing theory may provide many ideas for potential sensors, we have also found it useful to examine the collected recordings to obtain new ideas for potential sensors. If a person can view the recordings and determine what the output of a statistical model should be, then ideas for potential sensors can be generated by examining what information the person is using to make their decision.

Systematically simulate potential sensors from the collected recordings. After deciding upon a set of potential sensors, we use a Wizard of Oz technique to systematically simulate each potential sensor. Working from a definition of each potential sensor, a human views the collected recordings and specifies the value that would have been output by the sensor if it were present. Importantly, this is not the same as specifying exactly what is taking place in an environment.

Instead, a sensor definition should indicate the expected limitations of the potential sensor and these limitations should be included in the simulation.

Learn statistical models from the simulated sensors and select potential sensors based on the utility of their simulated versions and their expected cost of implementation. Using the collected labels and the results of systematic Wizard of Oz sensor simulation, statistical models can be constructed and analyzed to determine which simulated sensors provide the most predictive value. For example, a potential sensor that would be very difficult to implement might be found to have very little predictive value. Or it may be the case that a much easier to implement sensor can provide most of the predictive value of a more difficult sensor. Using the knowledge gained by analyzing statistical models based on the simulated potential sensors, informed decisions can be made about what sensors to actually develop.

Implement the selected sensors. Several potential issues are inherent to any Wizard of Oz sensor simulation. The simulated sensor might be less noisy than an implemented version, the definition of a simulated sensor might not exactly match what can be implemented, or the person simulating a potential sensor might have unintentionally introduced a systematic error. Because of these potential issues, it is important to create actual implementations of sensors that are found to be predictive.

Validate the implemented sensors in the deployment environment. After implementing actual sensors, it is generally worthwhile to conduct deployments that are larger or provide additional insight into the deployment environment. We will discuss two such validation deployments. In the first, we implement sensors based on the results of analyzing audio and video recordings from normal office work environments. We then validate the implemented sensors using office workers with more diverse responsibilities and work environments. In the second, we implement software-based sensors based on the results of analyzing screen capture recordings of programmer development environments. We then validate the utility of low-level events in the development environment by studying a larger set of programmers.

While this section has only provided an overview of the seven steps in our approach, later chapters will present a detailed application of our approach to two problems. Chapters 4, 5, and 6 will discuss the use of environmental sensors to model the interruptibility of office workers in their normal work environments. Chapter 7 will focus on task engagement, using low-level events in a programming environment to develop a sensor-based statistical model of programmer interruptibility. Though we will not discuss it here, we have also begun to apply our approach to home activity recognition using unobtrusive and low-cost microphone-based sensors (Fogarty *et al.* 2006).

3.3 Benefits of Our Approach

Applying our approach to the top-down development of sensor-based statistical models has several benefits. This section explicitly discusses three: exposing the tradeoff between complexity and utility, enabling the iterative exploration of potential sensors, and inspecting the cause of errors in a sensor-based statistical model.

3.3.1 Exposing the Tradeoff Between Complexity and Utility

In a traditional bottom-up approach, a developer must rely on their intuition and on existing theory when deciding what sensors to implement and deploy. Both intuition and theory are important, but they are often based on relatively qualitative observations and it is generally hard to translate them into accurate quantitative estimates of a sensor's utility in a statistical model. Anecdotally, it can be easy to fall victim to the belief that a more complex sensor, because it considers more of the environment, will be more useful in a statistical model. This is illustrated in Figure 3.2, showing several hypothetical sensors plotted according to the complexity of their implementation. Without additional information, it is easy to believe that Sensor F will be the most predictive.

Our top-down approach to sensor-based statistical models exposes the tradeoff between a sensor's complexity and its expected utility in a statistical model. This is illustrated in Figure 3.3 by plotting the complexity of a sensor against its utility in a sensor-based statistical model. In this hypothetical example, Sensor C provides very high utility relative to its complexity and Sensor F provides very low utility relative to its complexity. It would almost certainly be inappropriate to spend the time and resources necessary to develop Sensor F. A more interesting question is whether to develop Sensor E, which provides the highest utility but is fairly complex. Depending on the

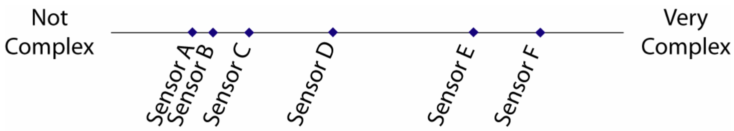


Figure 3.2. Without information about a sensor's utility, decisions can be based only on the complexity of a sensor.

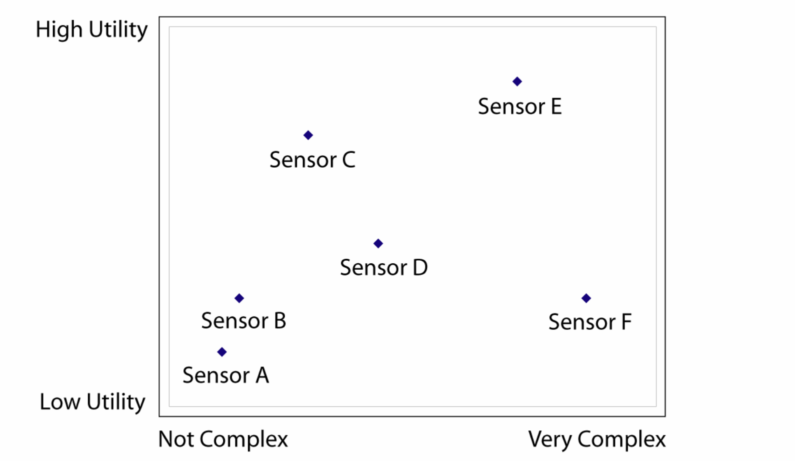


Figure 3.3. Our approach allows the consideration of both a sensor's complexity and it expected utility.

application for which a sensor-based statistical model is being developed, the relatively simple Sensor C may be the best choice. It provides most of the utility of Sensor E with a much lower complexity.

3.3.2 Enabling the Iterative Exploration of Potential Sensors

In a traditional bottom-up approach, sensors must be developed and deployed before they can be evaluated. If the evaluation shows that the developed sensors are inadequate, it is typically very expensive to recover. While a researcher may create an additional sensor to address a shortcoming discovered in the evaluation, an entirely new evaluation must then be conducted to determine if the new sensor successfully addresses the issue.

In contrast, using a Wizard of Oz technique in our top-down approach enables relatively low-cost iterative exploration of potential sensors. If the initial set of simulated sensors is found to be ineffective, the collected recordings can often be used to simulate and examine an additional set of potential sensors. Because the recordings have already been collected, the cost of using our approach is much lower than the cost of collecting new data at each point in an iterative process. Rapid iteration is a tenet of successful approaches to human-computer interaction, and our approach enables rapid iteration in the development of sensor-based statistical models.

3.3.3 Inspecting Errors in a Sensor-Based Statistical Model

As noted earlier in this chapter, the success of a sensor-based statistical model includes two distinct requirements: it must be based on sensors that reliably detect appropriate context *and* the sensed context must have a meaningful statistical relationship to the concept being modeled. If an evaluation shows that a sensor-based statistical model is unreliable, a traditional bottom-up approach makes it difficult to determine whether errors are caused by noise introduced as a result of a shortcoming in the sensor implementation or by the absence of a meaningful statistical relationship with the concept being modeled.

Because our approach uses a Wizard of Oz technique to simulate potential sensors, the simulated version of a sensor provides its ideal output. By building statistical models from the simulated ideal output of potential sensors, we determine the degree to which a highly-reliable sensor is useful in a statistical model. The error rate of a model based on simulated ideal sensors provides a target error rate for models based on actual, implemented sensors. Once sensors have been implemented and evaluated, the difference between the target error rate and the actual error rate provides an estimate of the degree to which sensor reliability is interfering with the sensor-based statistical model. By differentiating these two sources of error in a sensor-based statistical model, our approach allows a greater understanding of how to go about improving a model.

3.4 Conclusion

This chapter has introduced our top-down approach to developing sensor-based statistical models. Whereas traditional bottom-up approaches are based in developing, deploying, and evaluating sensors, our approach uses a Wizard of Oz technique to examine the utility of simulated versions of potential sensors. Taking our top-down approach exposes both dimensions of a tradeoff between sensor complexity and sensor utility, enables the iterative exploration of potential sensors, and allows an understanding of whether errors are caused by a shortcoming in a sensor implementation or the absence of a statistical relationship between a sensor and the concept being modeled. The coming chapters apply our approach to examine sensor-based statistical models of human interruptibility.

Chapter 4

A Wizard of Oz Study of Office Worker Interruptibility

4.1 Introduction

Sensor-based statistical models of human interruptibility are one approach to addressing the human costs of systems that interrupt inappropriately or unduly demand attention. Because people use social conventions and externally visible cues to estimate interruptibility, rather than relying on invisible internal phenomena like cognitive state, it should be possible to develop such models empirically. One method would be a bottom-up approach, based on the development, deployment, and evaluation of sensors and models. However, the uncertainty surrounding the usefulness of various sensors makes it very likely that significant time and resources would be spent building and evaluating sensor ill-suited or suboptimal for a model of human interruptibility. This work is instead based on a top-down approach, in which we collect and analyze more than 600 hours of audio and video recordings from the actual working environments of four participants with no prior relationship to our research group. We simultaneously collect self-reports of their interruptibility. Using a Wizard of Oz technique, as outlined in the previous chapter, we simulate the presence of a variety of potential sensors and create models based on the assumption that changes in behavior or context are indicative of interruptibility. We show that models based on simulated sensors can identify "Highly Non-Interruptible" situations with an accuracy of 79.2% and an A' of .850.

This chapter starts by presenting our data collection. We then summarize the collected interruptibility self-reports and the sensed context surrounding those reports. We next discuss our Wizard of Oz simulation of many potential sensors. After this, we develop a set of potential features based on the output of our simulated sensors. We then use the potential features to build statistical models. Finally, we evaluate the accuracy of the resulting models and examine several questions surrounding which features are most useful in a statistical model.

4.2 Data Collection

We collected recordings in the actual working environments of four participants with no prior relationship to our research group. To increase uniformity for this exploratory work, we selected four participants with similar working environments and tasks. Each participant serves in a high-level staff position in our university with significant responsibilities for day-to-day administration of a large university department and/or graduate program. The participants have private offices with closable doors, but their responsibilities require them to interact with many different people and they generally do not have full control over their time. They usually work with their doors open and respond to a variety of walk-in requests. Because they almost never close their office doors, it is likely that the absence of this explicit indication of non-interruptibility makes it more difficult to estimate their interruptibility.

Recordings were captured using a computer with an 80GB disk and an audio/video capture card connected to a small camera and microphone. Participants could disable recordings for thirty minutes by pressing the space bar on the machine collecting the recordings. The computers had speakers used for informing participants that recording was disabled, to advise them that recording was about to resume, and to request interruptibility self-reports. While we used a display during the initial configuration and installation of the recording machine, the machines did not have displays during data collection. Signs were posted to alert guests to the presence of a recording device, and the participants were encouraged to disable recording if they or a guest was uncomfortable. We also provided participants with a mechanism for retroactively requesting that recordings be destroyed before viewing.

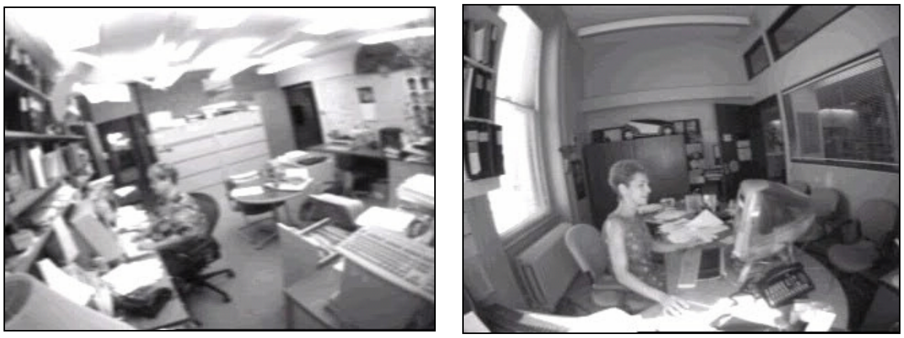


Figure 4.1. Two representative frames from our recordings.

Grayscale cameras with wide-angle lenses were mounted in the office so that both the primary working area and the door were visible. Figure 4.1 shows representative images from two of the cameras. Video was captured at approximately 6 frames per second, at a resolution of 320x240. Audio was captured at 11KHz, with 8-bit samples. The recording machines were deployed for between 14 and 22 workdays for each participant, recording from 7am to 6pm on workdays. Our setup worked well except in one case where a week of data was lost because an undetected improper compression setting caused the disk to fill prematurely. For this participant, we collected an additional 10 days of data at a later date. A total of 602 hours of recordings was collected from the offices of these four participants.

Participants were prompted for interruptibility self-reports at random, but controlled, intervals averaging two prompts per hour. This is an experience-sampling technique, sometimes known as a beeper study (Larson and Csikszentmihalyi 1983; Feldman-Barrett and Barrett 2001). To minimize compliance problems, we asked a single question rated on a five-point scale. Participants could answer verbally or by holding up fingers on one hand, but almost all responses were verbal. Participants were asked to “rate your current interruptibility” on a five-point scale, with 1 corresponding to “Highly Interruptible” and 5 corresponding to “Highly Non-Interruptible.” A sign attached to the recording machine reminded the participant which value corresponded to which end of the scale. Participants were present for a total of 672 of these prompts.

4.3 Collected Data Overview

This section characterizes the data collected from our participants. The overall distribution of interruptibility self-reports is shown in Figure 4.2. The distributions for

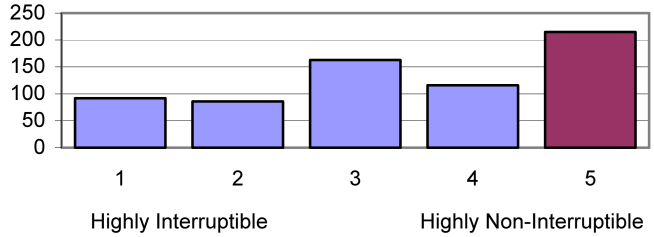


Figure 4.2. Distribution of interruptibility self-reports. We focus on distinguishing "Highly Non-Interruptible" responses from the other four possible responses.

	Highly Interruptible				Highly Non-Interruptible
	1	2	3	4	5
Subject 1	9 6.6%	14 10.2%	40 29.2%	18 13.1%	56 40.9%
Subject 2	17 10.2%	21 12.7%	58 34.9%	27 16.3%	43 25.9%
Subject 3	52 31.5%	26 15.8%	20 12.1%	10 6.1%	57 34.5%
Subject 4	14 6.9%	25 12.3%	45 22.1%	61 29.9%	59 28.9%
All	92 13.7%	86 12.8%	163 24.3%	116 17.3%	215 32.0%

Figure 4.3. Individual distributions of self-reports.

individual participants are show in Figure 4.3. For 54 of these 672 samples, the participant was present and clearly heard the prompt, but did not respond within 30 seconds. We examined these individually and determined that the participant was either on the phone or with a guest for the vast majority of the 54 cases. Results in the literature suggest that these activities are highly correlated with non-interruptibility, and this expectation is validated in the remainder of our data. To simplify analysis and model building, we have placed these 54 cases in the "Highly Non-Interruptible" category (a 5 on our five-point scale).

While there are clearly differences in the self-report distributions for the individual participants, it is especially important to note that the participants self-reported that they were "Highly Non-Interruptible" for 215 prompts, or approximately 32% of the data. An informal inspection found that responses of "Highly Non-Interruptible" were sometimes given calmly and other times curtly by agitated participants. This distribution seems to

Door Open	98.6%
Occupant Sit	88.9%
Occupant at Desk	74.0%
Occupant Keyboard	22.6%
Occupant Monitor	46.8%
Occupant Papers	28.0%
Occupant Drink	1.0%
Occupant Talk	32.6%
One or More Guests Present	24.1%
One or More Guests Sit	9.3%
One or More Guests Stand	14.2%
One or More Guests Talk	20.7%
One or More Guests Touch	0.5%

Door Close	0.7%
Occupant Stand	13.1%
Occupant at Table	21.2%
Occupant Mouse	19.6%
Occupant File Cabinet	1.0%
Occupant Write	5.5%
Occupant Food	1.4%
Occupant on Telephone	12.7%
Two or More Guests Present	3.0%
Two or More Guests Sit	1.5%
Two or More Guests Stand	0.8%
Two or More Guests Talk	1.7%
Two or More Guests Touch	0.0%

Figure 4.4. Frequency of various events during times when the office occupant was present.

indicate that there are circumstances in which people consider themselves to be clearly non-interruptible, as opposed to other times when their interruptibility might be more dependent on the nature of the interruption or some other factor. We therefore focus on models that distinguish "Highly Non-Interruptible" situations from the other four possible values of a self-report. If models could reliably recognize the times when people are clearly non-interruptible, this would be a significant advance over current applications.

Figure 4.4 presents how often particular events occur in the collected recordings. These values are based on the manually simulated sensors discussed later in this chapter. They are also based on the periods for which the participant was present, as opposed to the entirety of the recordings. As previously mentioned, these participants almost always had their doors open and the lack of the explicit non-interruptibility cue provided by a closed door probably makes it more difficult to estimate their interruptibility. The participants spent most of the day sitting, and most of that time sitting at their desks. A guest was present approximately 25% of the time when the participants were present, but there was very rarely more than one guest present. While participants frequently interacted with a computer, they also spent a significant amount of time handling papers or talking.

4.4 Wizard of Oz Sensor Simulation

Working from the collected recordings, we used a Wizard of Oz technique to simulate a number of potential sensors. We developed custom software for this purpose, shown in Figure 4.5. The interface presents recordings in 15-second segments. A coder could

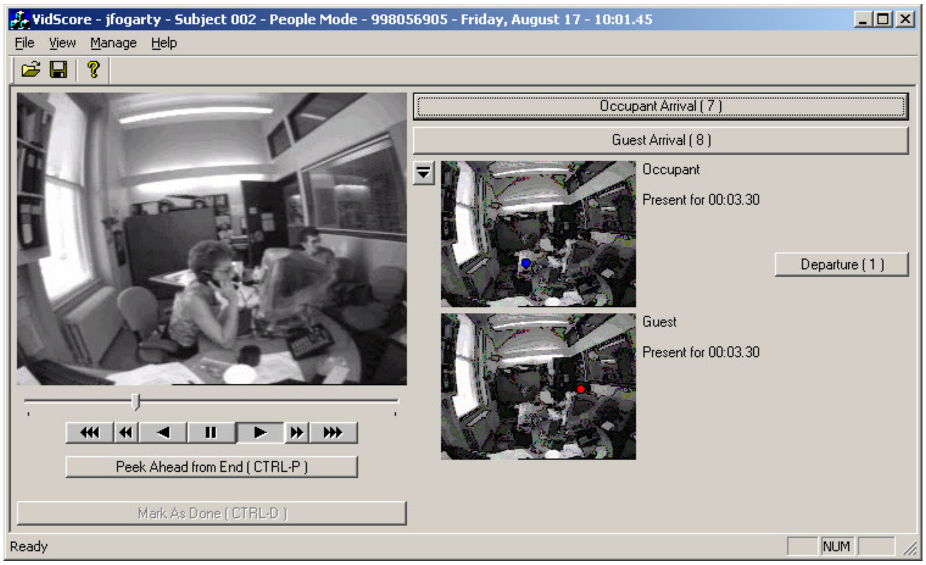


Figure 4.5. The custom interface used
in our Wizard of Oz simulation of potential sensors.

playback the recordings at normal speed or double speed, at their option. At the end of each segment, a coder could go to the next segment or watch the current segment again. For each of our simulated sensors, the person viewed the recordings and indicated whether the event occurred at any point in the 15-second segment.

We developed this interface and the set of sensors it simulates after an initial exploratory coding of the video from our first participant. In this initial exploratory coding, our video coders viewed the recordings and coded precise start and end times for any activities they thought were potentially relevant. While this resulted in many ideas for potential sensors, the precise coding of start and end times is very time consuming and the resulting set of coded activities was inconsistent both between and within coders. In contrast, our final interface provided a well-defined set of relevant events and appropriate keyboard shortcuts allowed coders to quickly indicate whether a particular event occurred at any point in a 15-second window. This use of a 15-second window is similar to the methods used in Lag Sequential Analysis (Sackett 1978; Osofsky 1979). Because we were interested in a large number of potential sensors, we broke our set of potential sensors into four groups and coded each group in a separate pass through the recordings. This reduced the cognitive demands on the coders and allowed us to automatically skip the coding of some sensors for some segments of the recordings. For example, if an early pass indicated that no guests were present during a segment, our interface automatically skipped that segment during the pass for coding guest activities.

We also minimized coding time by coding only the 5 minutes of video preceding each self-report (a total of 56 hours of coded recordings), as we believe information in close temporal proximity will be most useful in predicting interruptibility.

Using detailed definitions of potential sensors, we simulated the detection of a total of 24 events or situations. Briefly summarized, these are:

- Is the office occupant present.
- Is the office occupant sitting, standing, speaking, or on the phone.
- Is the office occupant touching or interacting with their desk (primary work surface), table (large flat surface other than the primary work surface), file cabinet, food, drink, keyboard, mouse, monitor (gaze at), papers (including books, newspapers, and loose paper), or writing instruments.
- How many guests are present.
- Is each guest sitting, standing, speaking, or in physical contact with the office occupant (any physical contact or close physical proximity with the occupant, including handing the occupant an object).
- Is the door open, is the door closed (when ambiguous, neither event was coded).
- Time of day (at an hour granularity).
- Is anybody talking (the logical OR of the office occupant speaking and any guests speaking).

We chose these manually simulated sensors because we had an *a priori* belief that they might relate to interruptibility, because we believed that a sensor could plausibly be built to detect them, and because they could be observed in our recordings. While we believed that information like what applications are running on a computer could be useful, we could not directly observe such information in our recordings. Some sensors would be easier to build than others, and we have included sensors that would be difficult to build because knowing that they are useful might justify the effort necessary to develop them. For example, our anybody talking sensor would be much easier to build than a sensor that detects whether it is the office occupant who is talking, but simulating both of them allows us to evaluate the difference in their utility.

Data from all four participants was coded after finalizing the coding procedures. We evaluated agreement among coders by recoding a randomly selected 5% of the recordings. This found a 93.4% agreement on sensor simulation at our granularity of 15 seconds.

4.5 Potential Feature Development

In order to develop a set of potential features appropriate for use in a statistical model, we applied a set of functions to the output of our simulated sensors. These functions are intended to capture recency, density, and change effects in the value of a simulated sensor during the time preceding a self-report. The six functions we used are:

- **Imm**: Whether the event occurred in the 15 second interval containing the self-report.
- **All-N**: Whether the event occurred in *every* 15 second interval during the *N* seconds prior to the self-report.
- **Any-N**: Whether the event occurred in *any* 15 second interval during *N* seconds prior to the self-report.
- **Count-N**: The number of 15 seconds intervals in which the event occurred during the *N* seconds prior to the self-report.
- **Change-N**: The number of consecutive intervals for which the event occurred in one and did not occur in the other during the *N* seconds prior to the self-report.
- **Net-N**: The difference in the sensor between the first interval in the *N* seconds prior to the self-report and the sensor in the interval containing the self-report.

We applied these functions for time intervals of 30 seconds, 1 minute, 2 minutes, and 5 minutes. In the remainder of this chapter, we will use these function names to refer to features. For example, “Occupant Talk (Any-5 minutes)” refers to the application of the Any function with the Occupant Talk sensor over a 5 minute interval.

4.6 Statistical Model Development

This section discusses the development of statistical models based on the potential features created by applying our functions to the output of our simulated sensors. While the functions yield almost 500 potential features, using all of these in a statistical model could have very negative effects. Some may not have a relationship to interruptibility, and some may lead to a phenomenon known as overfitting. When a model is overfit, is mistakenly interprets minor details or quirks in a feature as representative of data it will be asked to evaluate in the future. The overall accuracy of its future estimates is then lower than it should be, because it is confused by differences in the minor details that it previously mistook for important. Overfitting is very similar to degree-of-freedom problems found in models with excessive parameters.

In order to construct models from an appropriate set of features, we apply a wrapper-based feature selection technique (Kohavi and John 1997). In this approach, an optimization performs a heuristic search for an optimal subset of the potential features. Starting with an empty feature set, the optimization adds and removes features until no change to the set results in a better model. A limited backtracking facility is included, allowing the optimization to search more than a single path. Each potential set of features is evaluated using a standard ten-fold cross-validation. The data is divided into ten *folds*, and ten trials are conducted. Each trial trains a model from the data in nine folds and tests the model against the data in the remaining fold. The optimization seeks to maximize the resulting estimate of the area under the ROC curve, a measure that is related to accuracy but avoids several pitfalls of optimizing directly for accuracy (Hanley and McNeil 1982; Hand and Till 2001; Fogarty et al. 2005a).

The statistical models presented in this dissertation will be naïve Bayes classifiers, a relatively simple classifier that has proven effective in a variety of problem domains (Duda and Hart 1973; Langley and Sage 1994). We have previously examined the use of decision trees (Quinlan 1993), support vector machines (Burges 1998), and AdaBoost with decision stumps (Freund and Schapire 1997), but found no significant differences in their ability to model our data (Hudson *et al.* 2003; Fogarty *et al.* 2005b). We will not attempt to fully describe these techniques, but instead refer readers to the original references or a machine learning text, such as (Mitchell 1997).

Figure 4.6 presents the accuracy of a naïve Bayes model built from 20 features created by applying the functions discussed in this chapter to the simulated output of the potential sensors discussed in this chapter. This model distinguishes situations reported as “Highly Non-Interruptible” from situations reported as any other value on our five-point scale. The model’s accuracy is presented in a form known as a confusion matrix.

	Simulated Sensor Model	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	402 59.8%	55 8.2%
Highly Non-Interruptible	85 12.6%	130 19.3%
	Accuracy: 79.2% vs. 68.0% prior A': .848, z = 24.4, *p* < .001	

Figure 4.6. The reliability of a naïve Bayes model of human interruptibility based on our simulated sensors.

Because we will use confusion matrices throughout this dissertation, it is worth detailing exactly what information they present. The rows in this matrix correspond to the actual interruptibility of a participant, as measured in self-reports. The columns correspond to the predictions made by the model. The unshaded diagonal therefore indicates the cases where the model correctly determined whether a given situation was "Highly Non-Interruptible", while the shaded diagonal indicates model errors. This model has an overall accuracy of 79.2%, computed as (402 + 130) / (402 + 55 + 85 + 130). Looking at individual rows and columns, we can see that the model correctly identifies 88.0% of the situations in which participants were not "Highly Non-Interruptible", which is computed as 402 / (402 + 55). We can also see that, when the model predicts that a person is not "Highly Non-Interruptible", it is correct 82.5% of the time, computed as 402 / (402 + 85). Similar computations show that the model identifies 60.5% of "Highly Non-Interruptible" situations and that it is correct 70.3% of the time that it predicts a situation is "Highly Non-Interruptible." This dissertation will generally focus on the overall accuracy of models, but we note that this additional information is available in each confusion matrix. This information might be important in determining whether a model is appropriate for a particular application. For example, consider using this model to filter incoming notifications when a person is "Highly Non-Interruptible." The model correctly identifies 88.0% of situations where the person is available, so 12% of notifications would be unnecessarily delayed. The model also identifies 60.5% of "Highly Non-Interruptible" situations, so 39.5% of inappropriate interruptions would still be allowed.

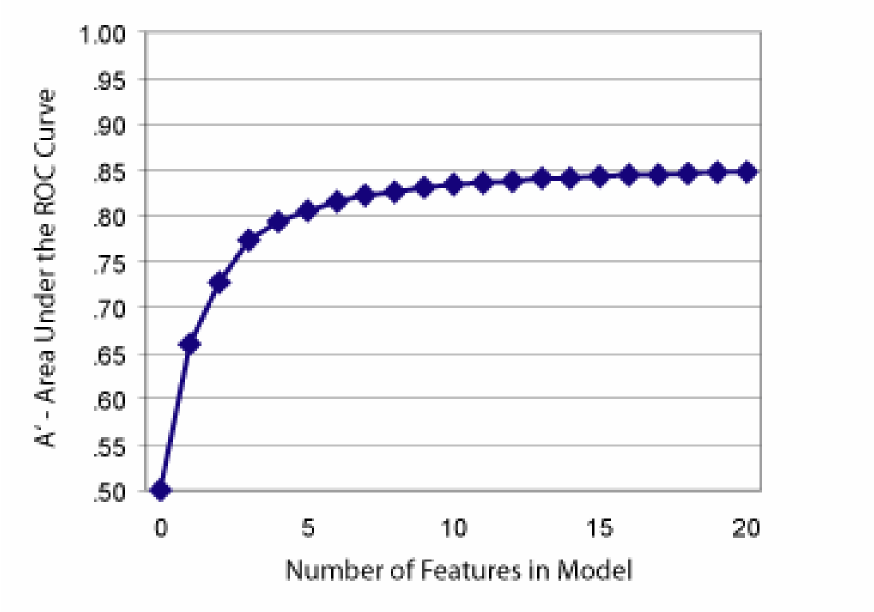


Figure 4.7. The model's reliability, indicated by A', jumps sharply with the first features and grows more slowly as additional features are added.

We will also provide a statistical test against the performance of a naïve estimator for each dataset in this document. A naïve estimator ignores context and always predicts the most common value in a particular dataset. In the case of the data shown in Figure 4.6, always predicting that a person is not "Highly Non-Interruptible" yields an accuracy of 68.0%, computed as (402 + 55) / (402 + 55 + 85 + 130). We note that this characterizes most current systems, which generally have no model and so assume that a person is always interruptible. We will refer to the performance of this naïve estimator as either the *prior* or the *chance* performance for a dataset, using the two terms interchangeably. The information necessary to compute A', the area under the ROC curve, is not available in a confusion matrix, but this document will also provide the value of A' for the data in each confusion matrix. The model in Figure 4.6 performs significantly better than the 68.0% chance performance for this dataset (A' = .848, z = 24.0, p < .001).

4.7 Examining the Selected Features

While the naïve Bayes model learned in the last section is based on 20 features, it gains much of its predictive power from a handful of features. Figure 4.7 illustrates this by plotting the increase in A', the area under the ROC curve, as additional features are added to the model. All zero-feature models have an A' of .500. The first feature added

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				68.0%	
.660	**Any Talk (Imm)**					
	No Talking	384	89	70.4%	82.2%	+14.2%
	Talking	73	126	29.6%	36.7%	-31.3%
.727	**Occupant Telephone (Count-30 seconds)**					
	< 30 Seconds	446	153	89.1%	79.8%	+11.8%
	>= 30 Seconds	11	62	10.9%	15.1%	-52.9%
.773	**Occupant Computer Mouse (Count-2 minutes)**					
	< 45 Seconds	294	185	71.3%	61.4%	-6.6%
	>= 45 Seconds	163	30	28.7%	84.5%	+16.5%
.793	**Occupant Papers (Any-5 minutes)**					
	Papers	285	143	63.7%	66.6%	-1.4%
	No Papers	172	72	36.3%	70.5%	+2.5%
.806	**Door Open (All-5 minutes)**					
	Open Entire 5 Minutes	289	131	62.5%	68.8%	+0.8%
	Ambiguous or Closed	168	84	37.5%	66.6%	-1.4%
.816	**Occupant Desk (Change-2 minutes)**					
	< 6 Changes	363	189	82.1%	65.8%	-2.2%
	>= 6 Changes	94	26	17.9%	78.3%	+10.3%
.822	**Occupant Table (Change-5 minutes)**					
	< 13 Changes	412	206	92.0%	66.6%	-1.4%
	>= 13 Changes	45	9	8.0%	83.3%	+15.3%

Figure 4.8. The first seven features selected for the simulated sensor model shown in Figure 4.6.

improves this to .660 and the second feature increases the model A' to .727. Later features still provide value, but their contribution is clearly smaller than the initial features. For example, the final feature improves the model A' from .847 to .848.

Figure 4.8 shows the first seven features selected, together with the relationship between values of these features and our collected interruptibility self-reports. The decision to present seven features is arbitrary, and it is clear that some of the later features are providing relatively little predictive value. We will always present seven features in similar figures throughout this document, but note that the first few features are typically the most predictive. In order to provide some indication of the importance of each feature, we present the value of A' as each feature is added.

The first feature selected is the Any Talk (Imm) feature, indicating whether anybody in the office was talking in the 15-second interval containing a prompt for a self-report. For 70.4% percent of self-reports, this feature has a value of false (nobody was talking). In this case, the likelihood that a person is not “Highly Non-Interruptible” is increased by

14.2%. On the other hand, somebody was talking for 29.6% of self-reports. For these self-reports, the likelihood that the participant is not "Highly Non-Interruptible" drops by 31.3%. This feature is therefore a fairly good indicator of interruptibility, leading the wrapper-based feature selection optimization to choose it as the most predictive individual feature.

After selecting the Any Talk (Imm) feature, the best feature to add is whether the participant has been on the phone at some point in both of the two most recent 15-second intervals. These first two features both capture strong cues related to social engagement, indicating that participants were less interruptible when a conversation was in progress or when on the phone. The third feature, whether the person has used the mouse during 3 or more 15-second intervals in the past 2 minutes, indicates that certain mouse-intensive computer activities are more interruptible. The fourth feature is related to task engagement, showing that people are less interruptible if they have handled papers in the past 5 minutes. By this point, features are making smaller contributions to the reliability of the model. Whether the door has been open for the entire preceding five minutes provides a small improvement to the model, as do features related to the intermittent use of a desk or table.

4.8 An Easy to Build Set of Sensors

Together with the fact that the first several features provide most of the predictive power of the model in the previous section, it is especially interesting to note that the first three features can be very easily implemented. The audio processing research community has developed a variety of features appropriate for identifying human speech in audio, such as those in (Lu et al. 2002). The use of a phone can be detected by software running on the telephone switch, by a small piece of hardware placed between the phone and the wall-jack, or by a magnetic reed switch that senses when a handset is off its hook. Simple software can detect when a person is using their mouse. These three sensors yield a model with an A' of .773, which in this case corresponds to an accuracy of 78.0%.

We therefore decided to investigate the reliability of a model based entirely on these relatively easy to build sensors. Figure 4.9 presents the reliability of a model based on only the Any Talk sensor, the Mouse sensor, the Keyboard sensor, the Telephone sensor, and the Time of Day sensor. Based on 12 features derived from these sensors, this model

	Easy to Build **Simulated Sensor Model**	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	408 60.7%	49 7.3%
Highly Non-Interruptible	95 14.1%	120 17.9%
	Accuracy: 78.6% vs. 68.0% prior A': .797, z = 17.6, $p < .001$	

Figure 4.9. The reliability of a naïve Bayes model of human interruptibility based on an easy to build set of sensors (Any Talk, Mouse, Keyboard, Telephone, Time of Day).

has an accuracy of 78.6% and an A', the area under the ROC curve, of .797. This is worse than the model built from the full set of simulated sensors (A' = .797 vs. A' = .848, z = 2.26, $p \approx .024$), but still significantly better than the 68.0% prior (A' = .797, z = 17.6, $p < .001$). In the next chapter, we compare the performance of human observers to both our full set of simulated sensors and this easy to build set of simulated sensors.

Figure 4.10 presents the initial features selected for the model based on simulated versions of our easy to build sensors. The first three features are the same as those selected when the full set of simulated sensors is available (see Figure 4.8). The fourth feature is whether or not the person was using the keyboard during the 15-second interval containing the self-report. We note that the relationship between keyboard usage and interruptibility is not what might have been expected. If keyboard usage indicated that these participants were engaged in a task, we might expect them to be less interruptible. But these participants are actually more interruptible when using the keyboard (just as the third feature shows that they are more interruptible when using the mouse). This might indicate that social engagement dominates the interruptibility of these office workers, or that their engaging tasks do not include the use of the computer. The final three features capture whether a conversation has started in the past 5 minutes, whether the office worker has been on the phone for the entire previous minute, and whether the office worker started using the computer's mouse in the past two minutes.

A'	**Value**	**# Interruptible**	**# Highly Non Interruptible**	**% of Data**	**% Interruptible**	**% Change**
.500	**No Features**				68.0%	
.660	**Any Talk (Imm)**					
	No Talking	384	89	70.4%	82.2%	+14.2%
	Talking	73	126	29.6%	36.7%	-31.3%
.727	**Occupant Telephone (Count-30 seconds)**					
	< 30 Seconds	446	153	89.1%	79.8%	+11.8%
	>= 30 Seconds	11	62	10.9%	15.1%	-52.9%
.773	**Occupant Computer Mouse (Count-2 minutes)**					
	< 45 Seconds	294	185	71.3%	61.4%	-6.6%
	>= 45 Seconds	163	30	28.7%	84.5%	+16.5%
.780	**Occupant Computer Keyboard (Imm)**					
	No Typing	335	187	77.7%	64.2%	-3.8%
	Typing	122	28	22.3%	81.3%	+13.3%
.785	**Any Talk (Net-5 minutes)**					
	Stopped Talking or No Change	434	180	91.4%	70.7%	+2.7%
	Started Talking	23	35	8.6%	39.7%	-28.3%
.787	**Occupant Telephone (All-1 minute)**					
	Not on Phone	451	168	92.1%	72.9%	+4.9%
	On Phone	6	47	7.9%	11.3%	-56.7%
.789	**Occupant Computer Mouse (Net-2 minutes)**					
	Stopped Mousing or No Change	410	200	90.8%	67.2%	-0.8%
	Started Mousing	47	15	9.2%	75.8%	+7.8%

Figure 4.10. The first seven features selected for the easy to build sensor model shown in Figure 4.9.

4.9 Discussion

This chapter has presented a Wizard of Oz study examining the feasibility of sensor-based statistical models of human interruptibility. Using an experience sampling method and 600 hours of audio and video recordings, we showed that a model based on simulated sensors can identify “Highly Non-Interruptible” situations with an accuracy of 79.2% and an A' of .848. If used to filter notifications delivered at inappropriate times, this model could prevent 60.5% of inappropriate notifications while still allowing the prompt delivery of 88.0% of appropriately-timed notifications.

While it was initially unclear that human interruptibility could be reliably modeled, this work has shown that relatively easy to build sensors are appropriate for modeling human interruptibility. We have examined the simulated versions of an easy to build set of sensors, including sensors to detect whether anybody in an office is talking, whether an office worker is using the phone, the level of mouse and keyboard activity, and the time of day. These sensors can be deployed using existing work on recognizing human speech in audio (Lu *et al.* 2002), software on a phone switch or physical sensors to detect when a phone is in use, and simple software to detect mouse and keyboard activity. The simulated versions of these sensors support a model with an accuracy of 78.6% and an A' of .797, appropriate for filtering 55.8% of inappropriate notifications while still allowing the prompt delivery of 89.3% of appropriately-timed notifications.

Several results from this chapter shape the approach we pursue in the remainder of this dissertation. First, a sensor to detect whether anybody in an office is talking is clearly important to sensor-based models of office worker interruptibility. Later chapters will examine this finding in the context of implemented sensors and office workers with more diverse responsibilities and work environments. Second, many potential sensors do not seem to provide enough benefit to warrant the time and cost required to develop, install, and maintain them. Our easy to build sensor set provided an accuracy of 78.6%, while the full sensor set provided an accuracy of 79.2%. The difference in the area under the ROC curves for these two models is statistically significant, but probably does not justify the installation of cameras or expensive sensing infrastructure. The remainder of this dissertation is therefore focused on easy to build and low-cost approaches to deploying sensor-based statistical models of human interruptibility. Finally, it was unclear whether the accuracy of these models was good enough to warrant the use of models in applications. The next chapter therefore examines the reliability of human observers deciding whether the office workers studied in this chapter are "Highly Non-Interruptible."

Limitations of the work presented in this chapter suggest several areas for potential future work. Because this chapter is based entirely on sensors simulated from audio and video recordings, there are some potential sensors that we could not examine. For example, information about what application a person is using on their computer seems like it could be useful in a model of interruptibility, but this information is not visible in our collected recordings. While we implement and examine this specific sensor later in this dissertation, this limitation prevents us from examining potential interactions between this sensor and other potential sensors that we do not later implement. This could be common issue in applying our Wizard of Oz technique, so it seems worth examining hybrid approaches that include actual implementations of sensors that are easy to develop while using our Wizard of Oz approach to consider sensors that will be more costly or time-consuming to develop.

While this chapter and several that follow use self-reports as a ground truth measure of interruptibility, there is a significant opportunity for research to examine whether people can reliably report their own interruptibility. The potential for this concern is motivated by significant evidence that self-reports can be influenced by a variety of factors. The actor-observer effect shows that people are more likely to attribute their own actions to situational factors and other people's actions to personal or dispositional factors (Jones and Nisbett 1971). This could manifest in interruptibility self-reports as a bias for participants to describe their interruptibility in terms of their environment. Because our models are based on the environment, this bias in participant self-reports could inflate the estimated reliability of our models. People also tend to be over-confident in assessing the reliability of social predictions (Dunning et al. 1990), so people might be expected to be overly confident in predicting how they would respond to an interruption. Shrauger and Osberg show that self-reports of psychological assessments are less reliable than judgments by other people, potentially due to a conflict with how a people perceive themselves (Shrauger and Osberg 1981). In a more humorous example of biases in how people perceive themselves, two separate studies have shown that over 90% of people surveyed report having an above average sense of humor (Allport 1961; Lefcourt and Martin 1986). There is a possibility that our interruptibility self-reports were influenced by how participants perceive themselves. If a participant feels they are a "busy" or "important" person, they might under-estimate their interruptibility. Conversely, a person who believes they are "accessible" might over-estimate their interruptibility. This dissertation uses interruptibility self-reports as a ground truth measure, deferring examination of these potential issues. Convincingly demonstrating the validity of interruptibility self-reports is large enough of a research question that we will defer further discussion of it until our comments on future work in Chapter 9.

Finally, we have examined models that identify situations reported as "Highly Non-Interruptible", but this dissertation does not examine models that attempt to estimate interruptibility along the entire five-point scale used by participants. We have previously conducted limited analyses based on the full five-point scale (Hudson *et al.* 2003; Fogarty *et al.* 2005b), but several additional analyses could be conducted. For example, it might be interesting to examine models based on normalizing an individual's self-reports.

Chapter 5

A Study of Human Observers Estimating Office Worker Interruptibility

5.1 Introduction

While our initial Wizard of Oz study demonstrated the feasibility of sensor-based statistical models of human interruptibility, it was unclear what level of reliability is required before it is appropriate to include models in applications. In particular, it would be inappropriate to expect 100% recognition of human interruptibility, as even people clearly cannot make such a determination with perfect accuracy. But despite this limitation, people are generally able to interrupt others without being perceived as rude or intrusive. Human performance therefore seems to be a good benchmark for evaluating the performance of sensor-based statistical models of human interruptibility. Even though such models would sometimes make errors, applications could use them to help avoid situations where an interruption is obviously inappropriate. This would be a significant improvement over current applications, which generally assume that a person is always interruptible.

This chapter explores the reliability required of sensor-based statistical models of human interruptibility by examining the reliability of human observers making similar estimates. We show that human observers, using snippets of the recordings collected in the previous chapter, can identify "Highly Non-Interruptible" situations with an accuracy of 76.9%. Our statistical model based on simulated sensors, presented in Figure 4.6, therefore performs significantly better than the human observers (A' = .848 vs. A' = .720, z = 6.78, *p* < .001). Similarly, the model based on easy to build sensors, presented in Figure 4.9, also performs significantly better than the human observers (A' = .797 vs. A' = .720, z = 3.74, *p* < .001).

We start by presenting our methodology. We then informally discuss the strategies that observers reported when estimating office worker interruptibility. We next present the results of the human observer estimates, showing that human observers identify "Highly Non-Interruptible" situations with an accuracy of 76.9%. Before concluding, we discuss potential questions surrounding observer confidence in their estimates and the length of the recording snippet used to make each estimate.

5.2 Methodology

Using a website that advertises experiments conducted at our university, we recruited 40 human observers, each of whom was paid for a session that was scheduled to last one hour. A majority of our human observers were students at our university or at another university within walking distance. To protect the privacy of the office workers in the recordings, the human observers were shown still images of the office workers and asked if they recognized any of the office workers. They were only shown recordings of office workers that they did not recognize.

Each session started with an explanation of the task. The human observers were told to evaluate the recordings as if they were walking into that situation and needed to decide how interruptible the office worker was prior to deciding whether to interrupt. A practice portion was started, and the experimenter introduced the human observer to the interface shown in Figure 5.1. The interface presented five initially unchecked radio buttons for each estimate. The human observers were told that they could watch the video more than once and they were advised to be as accurate as possible without concern for speed. The human observers then used the interface to estimate the interruptibility of an office worker for 6 randomly selected practice self-reports. This was followed by the main portion, in which the human observer estimated the interruptibility of office workers for 60 self-reports. The main portion self-reports were selected randomly without replacement between human observers, ensuring that every self-report would be used

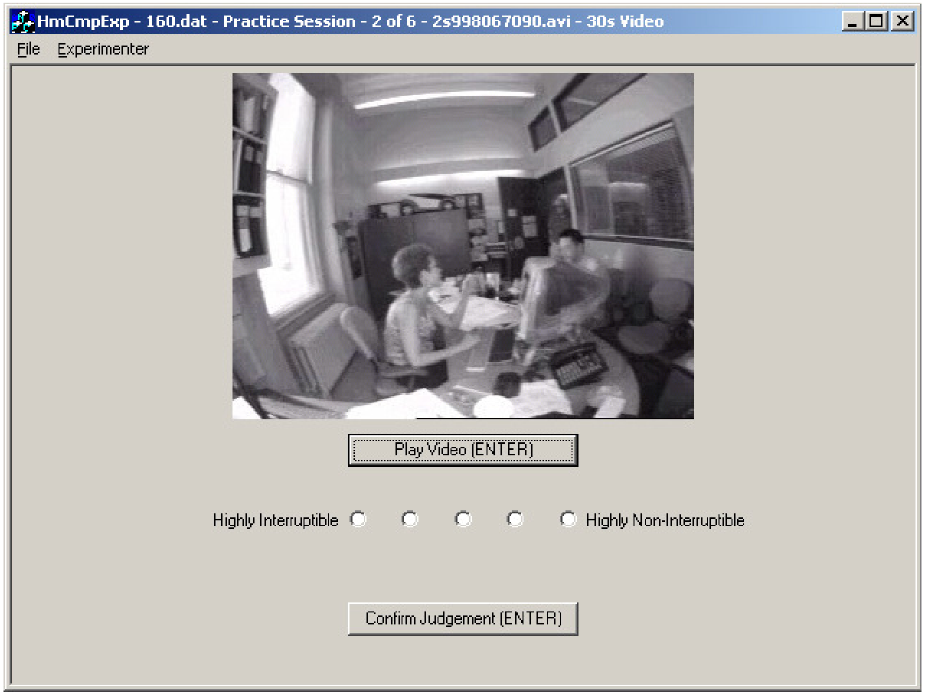

HmCmpExp - 160.dat - Practice Session - 2 of 6 - 2s998067090.avi - 30s Video
File Experimenter
Play Video (ENTER)
Highly Interruptible ○ ○ ○ ○ ○ Highly Non-Interruptible
Confirm Judgement (ENTER)

Figure 5.1. The custom interface used by our human observers to estimate the interruptibility of the office workers.

once before any self-report was used twice. After the main portion was completed, the human observers provided information about their general strategies during the main session and their specific strategies for making estimates from specific recordings. We finally collected responses to two seven-point Likert scales discussed later in this chapter. The sessions were not timed, but none lasted longer than the scheduled hour.

During both the practice and main portions, the interface alternated between showing 15 or 30 seconds of the recordings from immediately before a self-report. Half of the estimator participants started with 15 seconds, and half started with 30 seconds. We chose to use 15 seconds of the recordings because people naturally make these estimates very quickly. For example, a person glancing in an open office door can usually decide whether it is appropriate to interrupt. We felt that showing too much of the recordings for each estimate might affect how the human observers made their decisions. While it would normally be considered inappropriate to look in an open office door for 15 seconds, we felt that the additional temporal information presented in 15 seconds should help to correct for differences between normal circumstances and our recordings. The 30-second condition was included to determine whether additional time improved accuracy. As we will discuss later in this section, our human observers felt 15 seconds were sufficient and their performance did not improve with the longer recordings.

Of the original 672 interruptibility self-reports, recordings for 587 self-reports were used with the human observers. The others were not used because their content was potentially sensitive or because a technological artifact, such as a short gap in the video, might have been distracting to the human observer. As 40 participants each provided estimates for 60 self-reports selected randomly without replacement, each of the 587 self-reports had four or five estimates generated for it, including at least two based on 15 seconds of the recordings and at least two based on 30 seconds.

5.3 Human Observer Strategies

Our human observers reported strategies that are consistent with our intuition and the available literature suggesting that social and task engagement are important (Seshadri and Shapira 2001). We will not attempt to analyze the reported strategies, but instead report informal observations about commonly reported strategies.

The presence of a person other than the occupant of an office was reported as important in many strategies. As one observer commented, “You can’t just barge in on someone else’s appointment time.” Similarly, our human observers reported that they did not want to interrupt an office worker who was on the phone. In regard to both the presence of another person and the office worker being on the phone, our human observers generally reported that they attempted to determine if a conversation was work-related or of a more social nature. Our human observers reported feeling it was more appropriate to interrupt conversations of a social nature.

Our human observers also reported cues related to the environment and activities of the office worker. Several observers reported checking whether the door was open or closed. Many reported feeling that an office worker was more interruptible if music was playing. Reports indicated that it was difficult to estimate the interruptibility of an office worker who was using a computer or reading, but the observers also indicated that they felt less comfortable interrupting an office worker who was typing quickly, as opposed to occasionally typing or using only the mouse. This related to observer reports that they attempted to interpret the body language of the office worker, considering them more

		Human Observer Estimates				
		Highly Interruptible 1	2	3	4	Highly Non-Interruptible 5
Office Worker Self-Reports	1	172 7.2%	92 3.8%	41 1.7%	31 1.3%	10 0.4%
	2	94 3.9%	110 4.6%	72 3.0%	36 1.5%	5 0.2%
	3	150 6.3%	204 8.5%	133 5.5%	79 3.3%	45 1.9%
	4	82 3.4%	110 4.6%	116 4.8%	73 3.0%	39 1.6%
	5	89 3.7%	121 5.0%	101 4.2%	145 6.0%	250 10.4%
		Overall Accuracy: 30.7% Accuracy Within 1: 65.8%				

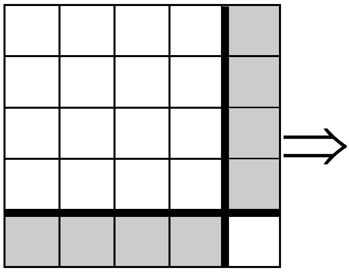
Figure 5.2. Confusion matrix for our human observer estimates of office worker interruptibility.

interruptible if they appeared to be happy or relaxing and less interruptible if they appeared stressed or upset.

Finally, a number of human observers reported that they felt more comfortable interrupting when an office worker was between tasks, such as when a guest had just left the office. The observers reported feeling that the office worker was already interrupted in this situation, so their interruption would not be problematic. But the observers also felt that they should not interrupt when an office worker was quickly switching back and forth between attending to two objects, as they felt this indicated the office worker was focused on a task.

5.4 Human Observer Estimate Reliability

Figure 5.2 presents the estimates made by our human observers in a confusion matrix. Rows correspond to the self-reports provided by our office workers, and columns correspond to the estimates provided by our human observers. Summing the diagonal, we can see that the human observers were correct for 738 instances, or approximately 30.7% of the data. Because "Highly Non-Interruptible" is the most common value, always estimating that value establishes a chance accuracy of 706 correct, or 29.4%. The human observers therefore performed only slightly better than chance, a difference which is not significant ($\chi 2(1, 4800) = 1.01$, $p > .31$). This indicates that interruptibility estimation, as posed, is difficult for human observers.



	Human Observer Estimates	
Office Worker Self-Reports	All Other Values	HighlyNon-Interruptible
All Other Values	1595 66.5%	99 4.1%
Highly Non-Interruptible	456 19.0%	250 10.4%
	Accuracy: 76.9% vs. 70.6% prior A': .720, z = 18.7, *p* < .001	

Figure 5.3. Confusion matrix for our human observers identifying "Highly Non-Interruptible" situations.

We note that the mistakes made by the human observers appear to include a certain amount of bias, perhaps related to self-interest. If the mistakes were random, we might expect approximately the same number of entries in the upper-right half of the confusion matrix as in the lower-left half. This would mean the human observers were equally likely to confuse office workers for being more interruptible as they were to confuse office workers for being less interruptible. Instead, there are 450 entries in the upper-right half, approximately 18.7% of the data, and 1212 entries in the lower-left half, approximately 50.5% of the data. Aggregating for each human observer, the human observers reported significantly lower values than the office workers (t(39) = -8.79, *p* < .001). This may imply a systematic bias towards viewing another person as interruptible when we are interested in making an interruption, a finding consistent with Avrahami *et al.*'s study of cell phone interruptions (Avrahami et al. 2006).

Figure 5.3 illustrates a transformation that reduces the problem to distinguishing between "Highly Non-Interruptible" situations and other situations. The bottom-right cell represents instances where both the office worker and the human observer responded with "Highly Non-Interruptible." The upper-left cell represents instances in which both the office worker and the human observer responded with any other value. The other two cells represent instances when either the office worker or the human observer responded with "Highly Non-Interruptible," but the other did not. For this problem, the human observers have an accuracy of 76.9%, significantly better than the chance performance of 70.6% (A' = .720, z = 18.2, *p* < .001).

While an accuracy of 76.9% may seem low for a task very similar to everyday tasks, we find this level of accuracy believable because of the context in which people normally

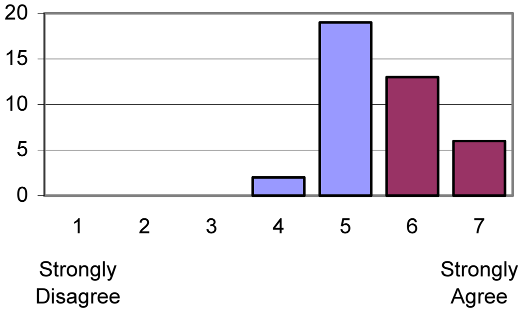


Figure 5.4. Human observer responses to the Likert scale "I am confident in the accuracy of my judgments."

make interruptibility estimates. As noted in the introduction to this chapter, people do not typically make an interruptibility estimate and then blindly proceed. Instead, the evaluation of interruptibility is an early step in a negotiated process (Kendon and Ferber 1973; Goffmann 1982). An initial determination that a person is not interruptible allows an early exit from negotiation, but other cues allow a person to decide against interrupting despite an initial evaluation that they could. Other cues can include eye contact avoidance and the continuation of the task that would be interrupted. In designing applications that use interruptibility estimates, it will be important to support a negotiated approach to interruptions, rather than assuming that interruptibility estimates provide absolute guidance.

5.5 Human Observer Confidence

The validity of our human observer results is strengthened by confidence data collected from the human observers. The first Likert scale in the experiment stated "I am confident in the accuracy of my judgments." Each human observer responded on a seven-point scale ranging from "Strong Disagree," which we will refer to as 1, to "Strongly Agree," which we will refer to as 7. Given the results for this scale, shown in Figure 5.4, it is clear that our human observers were confident in the accuracy of their estimates.

Furthermore, there is no significant difference in the reliability of estimates made by human observers who indicated greater confidence. The observers who indicated a confidence of 6 or 7 identified "Highly Non-Interruptible" situations with an accuracy of 76.1% and an A' of .724, versus an accuracy of 77.6% and an A' of .716 for observers

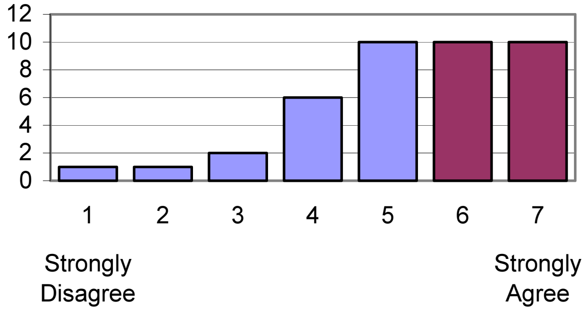


Figure 5.5. Human observer responses to the Likert scale "The 15 second videos were long enough for making judgments."

who responded with a confidence of 4 or 5. This difference is not significant (A' = .724 vs. A' = .716, z = 0.34, $p \approx .731$).

5.6 Recording Duration

As discussed in introducing our methodology, we felt 15 seconds of the recordings would be sufficient for estimating interruptibility and we included cases with 30 seconds to determine whether the additional time was helpful. This section presents evidence supporting our initial belief that 15 seconds of the recordings was sufficient.

The second Likert scale in the experiment stated "The 15 second videos were long enough for making judgments." Figure 5.5 shows the responses of our human observers, indicating that they generally found 15 seconds to be sufficient. There is no significant difference in the reliability of estimates made by observers who had more or less confidence in the use of 15 seconds of the recordings. The observers who responded with a value of 6 or 7 had an accuracy of 76.7% and an A' of .704, not significantly different from the observers who responded with a value of 5 or less and had an accuracy of 77.1% and an A' of .734 (A' = .704 vs. A' = .734, z = 1.22, $p \approx .224$).

Furthermore, there is no significant difference in the reliability of estimates based on 15 seconds of the recordings versus estimates based on 30 seconds of the recordings. When using 30 seconds of the recordings, observers had an accuracy of 77.1% and an A' of .716. When using 15 seconds of the recordings, observers had an accuracy of 76.7% and an A' of .724. This difference is not significant (A' = .716 vs. A' = .724, z = 0.33, $p \approx .741$).

5.7 Discussion

This chapter has presented an experiment to explore human observers estimating the interruptibility of office workers. We showed that human observers performed only slightly better than chance when asked to estimate office worker interruptibility on a five-point scale from "Highly Interruptible" to "Highly Non-Interruptible." These human observers also appear to have systematically interpreted the office workers as being more interruptible that the office workers reported. By reducing the problem to distinguishing between "Highly Non-Interruptible" conditions and other conditions, we establish an accuracy of 76.9% for human observers.

Taken with our results from the previous chapter, these results indicate that automated estimates of human interruptibility can be based on short periods of time immediately preceding a potential interruption. The fact that human observers have difficulty estimating office worker interruptibility on a five-point scale supports our decision to focus on sensor-based models that distinguish "Highly Non-Interruptible" situations from other situations. Our models could identify extremely inappropriate times for interruptions and allow a system to avoid them while using negotiated approaches during other times. This strategy appears to work well in human interaction (Kendon and Ferber 1973; Goffmann 1982) and also seems worth pursuing as an approach to human-computer interaction.

The performance of human observers in this chapter provides a strong point of comparison for the previous chapter's statistical models based on simulated sensors. The model based on our full set of simulated sensors, presented in Figure 4.6, identifies "Highly Non-Interruptible" situations significantly better than our human observers (A' = .848 vs. A' = .720, z = 6.78, $p < .001$). While the previous chapter showed that a model based on an easy to build set of sensors performed worse than a model based on the full set of simulated sensors, this chapter shows that the easy to build model still performs significantly better than the human observers in this study (A' = .797 vs. A' = .720, z = 3.74, $p < .001$).

Showing that our model based on easy to build sensors can identify "Highly Non-Interruptible" situations significantly better than human observers has two major implications for the work we pursue in the remainder of this dissertation. First, this provided more support for our decision to focus on sensors that can be developed, deployed, and maintained at a low cost. Second, the effectiveness of naïve Bayes models led us to focus on using relatively simple classifiers. While there is undoubtedly room to consider Markov processes or other more complex modeling techniques, this dissertation will focus on validating the effectiveness of our approach with more diverse office workers and developing software to enable the deployment of applications based on our approach. Given large datasets collected in real applications, we believe it would then be fruitful to examine a wider variety of potential learning algorithms.

The work presented in this chapter suggests a variety of future work to address limitations or explore additional questions. None of the human observers in this chapter had personal knowledge of the office workers whose interruptibility they were estimating, and it is possible that people personally familiar with an office worker may be able to better estimate the office worker's interruptibility. If this is the case, it would be interesting to examine how much of this is based on knowledge of a person's routines versus the actual context at the time of a potential interruption. It is also interesting to consider how different information about context, such as how many guests are present, influence interruptibility estimates made by human observers. Presenting different types of relationships to the human observer may also affect their estimates. While we told the human observers in this chapter to act as if they were approaching the office worker with an interruption, their estimates might change if they were told to act as an assistant regulating access to the office worker.

Chapter 6

A Robustness Study with Real Sensors and Diverse Office Workers

6.1 Introduction

While our initial studies demonstrated the feasibility of sensor-based statistical models of human interruptibility that perform better than human observers, several questions were left unanswered. The four office workers in the initial study all had similar jobs as high-level staff in our university, responsible for the day-to-day administration of a large university department and/or graduate program. It was unclear whether results obtained with these participants would generalize to different types of office workers, such as programmers or others who spend less time interacting with people. There was also a question of whether real sensors could be implemented reliably enough to obtain results as good as those obtained using simulated sensors. As noted in Chapter 3's discussion of our Wizard of Oz approach, it is important to address the potential that a simulated sensor could be less noisy than an implemented version, that the definition of a simulated sensors might not exactly match what can be implemented, or that the person simulating a potential sensor might have unintentionally introduced a systematic error.

This chapter presents a robustness study that deploys real, implemented sensors and examines sensor-based statistical models of the interruptibility of a more diverse group of office workers. We show that more reliable models are obtained when focusing on office workers with similar responsibilities and work environments, but that even a single model of our diverse participants still performs significantly better than the human observers from our previous study (A' = .838 vs. A' = .720, $z = 6.87$, $p < .001$). This shows that our approach to modeling human interruptibility applies to a wide variety of office workers, not just people in the original demographic. We also examine which sensors are most predictive for different types of office workers. Finally, we show that a typical laptop computer can support sensor-based statistical models that perform better than human observers (A' = .836 vs. A' = .720, $z = 5.27$, $p < .001$).

The next section presents an overview of our data collection, including an introduction to the office workers studied and a description of our deployed sensors. We then summarize our collected data, showing that the distribution of self-reports in this study is very similar to that in our original study. We then develop sensor-based statistical models that distinguish between "Highly Non-Interruptible" situations and other situations. After showing that a general model performs better than the human observers from our previous study, we present and analyze models of the interruptibility of office workers with similar responsibilities and work environments. We show that different sensors are most useful for different types of office workers, suggesting that applications should learn individualized models of human interruptibility. Finally, we examine models based on the sensing capabilities of a typical laptop computer, showing that software-deployable sensing is sufficient to support models that perform better than human observers.

6.2 Data Collection

We conducted this study in the offices of a major corporate research laboratory. We again use an experience sampling technique, which is sometimes referred to as a beeper study (Larson and Csikszentmihalyi 1983; Feldman-Barrett and Barrett 2001). After installing sensors in participant offices, we left and participants went about their normal work activities. At random intervals, our setup played an audio file prompting participants to report their current level of interruptibility.

6.2.1 Sensor Installation

Sensor data was collected by several background processes running on a participant's primary computer. Because some participants used laptop computers that they needed to be able to take away from their office, all of our sensors were attached to the computer by a single connection to a USB hub. Participants detached this connection when they took their laptop computer away from their office, and our software gave occasional subtle prompts for participants to reconnect the hub.

A set of speakers connected to the USB hub were used to prompt participants to give interruptibility self-reports. The speakers played an audio file asking the participant to "Please give your current interruptibility level." For the 10 seconds following each participant prompt, our software recorded audio from a microphone attached via the USB hub. During this time, participants responded orally on the same 5-point scale used in our prior studies (where a 1 indicates that a participant is "Highly Interruptible" and a 5 indicates that a participant is "Highly Non-Interruptible). A sign was posted in the participant's office to remind them which end of the scale corresponded to which value. At the end of the 10 seconds, a short tone was played to let participants know that the software was no longer recording audio. Participants were told that a non-response would be treated as a 5 if they were on the phone and could not answer. Non-responses when the participant was not on the phone were discarded, because there was no reliable way to determine whether the participant was present and had not responded or was just not present. We initially prompted participants at random intervals of between 40 and 80 minutes, but later increased the frequency of prompts to between 30 and 50 minutes. We made this change because it was common for participants to miss a prompt by stepping out of their office for only a few minutes.

Besides recording participant responses to the interruptibility prompts, the USB microphone was also used as a sensor to determine whether anybody was talking in the office. The microphones were placed on shelves in each office, about 8 feet from the floor and away from computer fans or other noise sources. The audio was analyzed in real-time on the participant's computer, using the silence detector provided by the CMU Sphinx speech recognition package (Huang et al. 1993). This software adapts to the relatively constant noise levels generated by fans, air conditioning, or quite background music. It identifies sharp increases in the energy of audio frames collected from the microphone, but does not indicates whether these sharp increases are talking or some other noise. However, conversations tend to go on for many seconds or even many minutes, whereas most other loud noises in an office environment are relatively short. In our experience, this system worked well for detected extended conversations. We used several different threshold configurations, logging the beginning and end times of non-silent intervals for each threshold configuration, but did not record audio. The analyses in this chapter are based on a count of how many threshold configurations were non-silent at a given point in time, so a silence detector value of 0 indicates that no threshold have been activated while a value of 22 indicates that every threshold has been activated. We later examine this sensor in a relatively noisy environment, offices in which more than one person normally works.

A custom-built USB sensor board was used to instrument each participant's office. Two magnetic switches, similar to those used in home alarm systems, were placed on either side of the top of the door frame. This allowed us to determine whether the door was open, cracked, or closed. Two motion sensors were put in each office, both about 5 feet above the floor, one near the door and one near the participant's desk. Another magnetic switch was used to determine whether a person's phone was physically off its hook. This physical switch could not detect if a person was using the speaker-phone functionality, but we were not allowed access to the phone systems that would have enabled reliable detection of this. In any case, the microphone-based talking sensor would be likely to detect talking when a participant used the speaker-phone.

Software on each participant's computer logged, once per second, the number of keyboard, mouse move, and mouse click events from the previous second. It also logged the title, type, and executable name of the active window and each non-active window. We chose to log this information out of the belief that some participants might be more or less interruptible when working in certain applications, a piece of information absent from the simulated sensors in our earlier study. All of the information associated with our study was automatically compressed and uploaded to a local server, so that we could verify that each participant's sensors seemed to be working properly and so that we could determine when each participant had provided an appropriate number of self-reports.

6.2.2 Participants

We collected sensor data and interruptibility self-reports from 10 participants with no prior relationship to this work. The participants were all employees of a major corporate research laboratory, studied during the course of their normal work. Of our 10 participants, two were first-line managers. We selected these participants because we felt that their human-centered work was closest to the responsibilities of the four participants

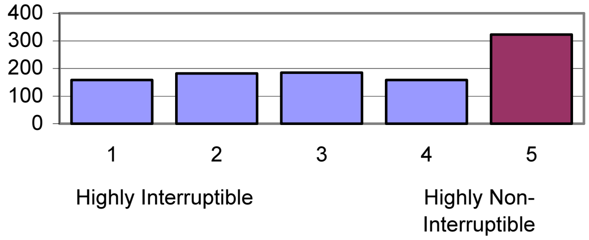


Figure 6.1. Distribution of interruptibility self-reports. We focus on distinguishing "Highly Non-Interruptible" responses from the other four possible responses.

from our original study. Just as social engagement, detected via the talking and phone sensors, was a key indicator of non-interruptibility in our initial study, we expected social engagement to be a key indicator for these two participants. Throughout this chapter, we shall refer to these two participants as the *managers*. Five of our 10 participants were researchers who spent a significant amount of their time programming. These participants were selected because they seem to represent typical knowledge workers, who do interact socially and attend meetings, but also work on tasks that require focused attention. Throughout this chapter, we shall refer to these five participants as the *researchers*. Finally, our last three participants were student interns. These participants also spent a significant amount of their time programming, but were selected because they shared an office with another intern. Because our initial studies indicated that a sensor to detect talking is very useful in a model of interruptibility, we included the intern participants to examine how the regular presence of a second person in the office affected the usefulness of our talk sensor implementation. While we would like to have studied the interruptibility of both people in shared offices, we were unable to find an office where both interns were willing to participate in the study. Throughout this chapter, we shall refer to these three participants as the *interns*.

6.3 Collected Data Overview

We set out to collect approximately 100 interruptibility self-reports from each participant. Figure 6.1 shows the distribution of the 1006 responses that were actually collected, while Figure 6.2 shows the responses from each individual participant. Data collection for participant 7, one of the researchers, was terminated early because of an external deadline that required the removal of the sensors from his office. Data collection for

		Highly Interruptible				Highly Non-Interruptible
#	Category	1	2	3	4	5
1	Manager	17 14.0%	31 25.6%	10 8.3%	19 15.7%	44 36.4%
2	Manager	2 2.0%	8 8.2%	36 36.8%	26 26.5%	26 26.5%
3	Researcher	67 62.0%	7 6.5%	8 7.4%	13 12.0%	13 12.0%
4	Researcher	23 22.3%	16 15.5%	27 26.2%	3 2.9%	34 33.0%
5	Researcher	4 3.7%	7 6.4%	24 22.0%	18 16.5%	56 51.4%
6	Researcher	5 4.5%	34 30.4%	29 25.9%	12 10.7%	32 28.6%
7	Researcher	0 0.0%	6 8.0%	12 16.0%	16 21.3%	41 54.7%
8	Intern	26 25.0%	28 26.9%	25 24.0%	11 10.6%	14 13.5%
9	Intern	3 4.6%	17 26.2%	12 18.5%	15 23.1%	18 27.7%
10	Intern	11 9.9%	28 25.2%	2 1.8%	25 22.5%	45 40.5%
All		158 15.7%	182 18.1%	185 18.4%	158 15.7%	323 32.1%

Figure 6.2. Individual distributions of self-reports.

participant 9, one of the interns, was terminated early because she expressed a feeling that the interruptibility prompts were annoying and asked for the sensors to be removed. Some participants went slightly over 100 responses because of the delay between the participant reaching 100 responses and our arrival to take down the sensors.

While there are individual differences in the distribution of interruptibility self-reports, we note that the most common response again was "Highly Non-Interruptible," accounting for 32.1% of the data. This distribution is very similar to the distribution found in our initial Wizard of Oz study, and seems to indicate that there are circumstances in which people consider themselves to be clearly non-interruptible, as opposed to situations where their interruptibility might be more dependent on the nature of the interruption or some other factor. The similarity of this distribution to the distribution in our original Wizard of Oz study also seems to validate our decision to focus on models that distinguish "Highly Non-Interruptible" situations from other

	All 10 Office Workers **Real Sensor Model**	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	575 57.2%	85 8.4%
Highly Non-Interruptible	129 12.8%	217 21.6%
	Accuracy: 78.7% vs. 67.9% prior A': .838, z = 27.6, *p* < .001 vs. Human Observers, z = 6.87, *p* < .001	

Figure 6.3. The reliability of a naïve Bayes model of human interruptibility based our actual, implemented sensors. This model is built and evaluated against data from all 10 office workers.

situations. The prior for this data is an accuracy of 67.9%, which could be obtained by always predicting that a person was not "Highly Non-Interruptible." As previously noted, this level of performance generally characterizes current systems, which lack a model of interruptibility and therefore always treat a person as interruptible.

6.4 A General Model of Office Worker Interruptibility

Figure 6.3 presents the reliability of a model of all ten office workers, based on the actual implemented sensors discussed earlier in this chapter. Built from 35 features, this model has an accuracy of 78.7% and an A' of .838. This is significantly better than the 67.9% chance accuracy for this data (A' = .838, z = 27.6, *p* < .001) and significantly better than the human observers in our previous study (A' = .838 vs. A' = .720, z = 6.87, *p* < .001).

This model and all of the other models presented in this chapter were developed in much the same way as the models in our initial Wizard of Oz feasibility study. Whereas Chapter 4 included a full list of the functions used to generate features, the models in this chapter were developed using `Subtle`, a tool that we have developed and will discuss in Chapter 8. For now, it suffices to say that `Subtle` applies an iterative feature generation algorithm to examine a variety of frequency, recency, and density features extracted from streams of sensor data. All of the evaluations presented in this chapter are based on standard ten-fold cross-validations, wherein the data is divided into ten folds. Ten trials are then conducted, each training a model with the data from nine of the folds. In each trial, the unused fold is used to test the resulting model.

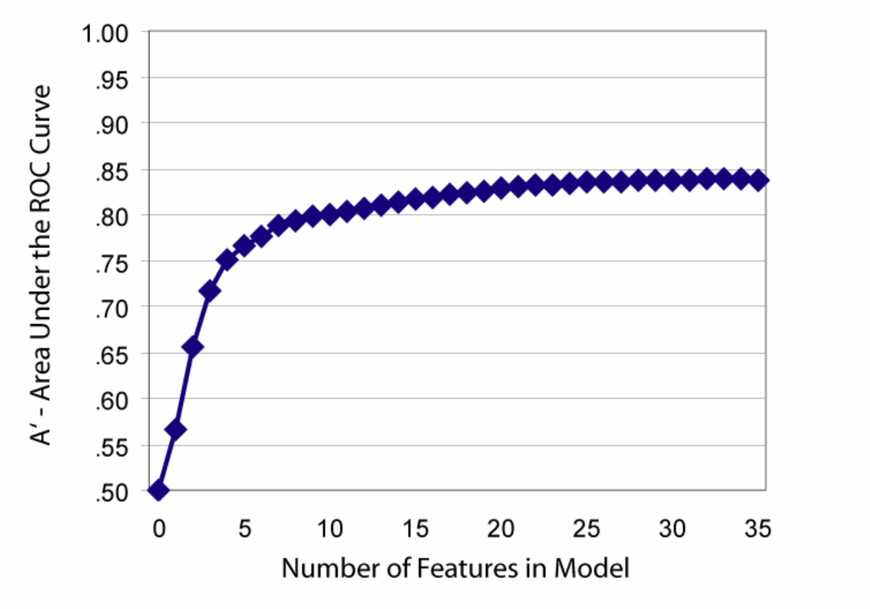
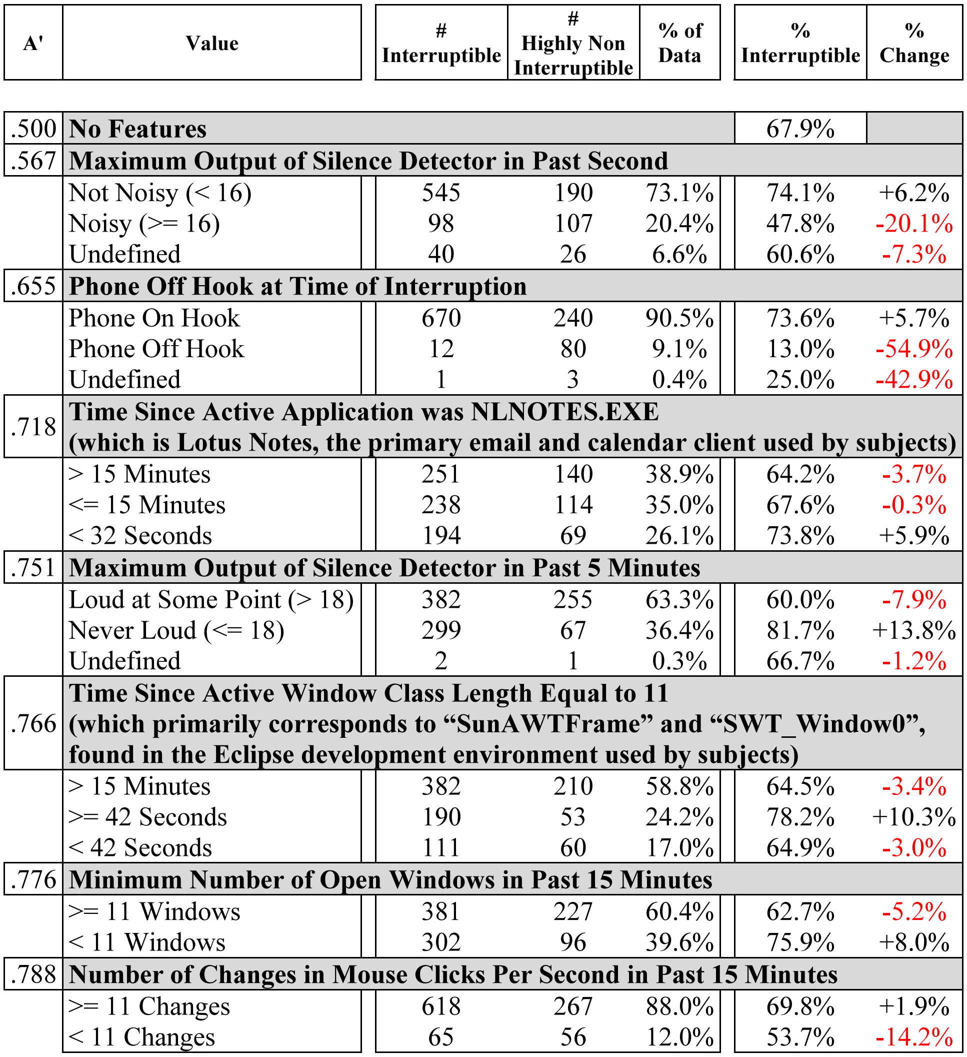


Figure 6.4. As in our initial Wizard of Oz feasibility study, the reliability of our model based on real sensors jumps sharply with its first features and grows more slowly as additional features are added.

As with the models developed in our initial Wizard of Oz feasibility study, much of the predictive power of this model comes from the first handful of features. Figure 6.4 plots the increase in this model's A' as features are selected. The first handful of features increase the model's A' from .5 to .566 to .655 and to .718.

Figure 6.5 shows the first seven features selected by this model. We note that occasional issues with the USB connection to our sensors resulted in small amounts of missing data, shown here as a value of "Undefined" for some features. This is more common with features that examine a very short time interval, as features that examine longer time intervals can recover from a short failure of the USB connection. Consistent with the results of our initial Wizard of Oz feasibility study, the first two sensors selected here are a threshold on the output of the microphone-based silence detector in the time immediately preceding an interruption and the physical sensor to detect whether the phone was off its hook at the time of a prompt for an interruptibility self-report. The selection of these sensors validates both our Wizard of Oz study, our implementation of these sensors, and the relevance of these sensors to modeling the interruptibility of a variety of office workers.



A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				67.9%	
.567	**Maximum Output of Silence Detector in Past Second**					
	Not Noisy (< 16)	545	190	73.1%	74.1%	+6.2%
	Noisy (>= 16)	98	107	20.4%	47.8%	-20.1%
	Undefined	40	26	6.6%	60.6%	-7.3%
.655	**Phone Off Hook at Time of Interruption**					
	Phone On Hook	670	240	90.5%	73.6%	+5.7%
	Phone Off Hook	12	80	9.1%	13.0%	-54.9%
	Undefined	1	3	0.4%	25.0%	-42.9%
.718	**Time Since Active Application was NLNOTES.EXE (which is Lotus Notes, the primary email and calendar client used by subjects)**					
	> 15 Minutes	251	140	38.9%	64.2%	-3.7%
	<= 15 Minutes	238	114	35.0%	67.6%	-0.3%
	< 32 Seconds	194	69	26.1%	73.8%	+5.9%
.751	**Maximum Output of Silence Detector in Past 5 Minutes**					
	Loud at Some Point (> 18)	382	255	63.3%	60.0%	-7.9%
	Never Loud (<= 18)	299	67	36.4%	81.7%	+13.8%
	Undefined	2	1	0.3%	66.7%	-1.2%
.766	**Time Since Active Window Class Length Equal to 11 (which primarily corresponds to "SunAWTFrame" and "SWT_Window0", found in the Eclipse development environment used by subjects)**					
	> 15 Minutes	382	210	58.8%	64.5%	-3.4%
	>= 42 Seconds	190	53	24.2%	78.2%	+10.3%
	< 42 Seconds	111	60	17.0%	64.9%	-3.0%
.776	**Minimum Number of Open Windows in Past 15 Minutes**					
	>= 11 Windows	381	227	60.4%	62.7%	-5.2%
	< 11 Windows	302	96	39.6%	75.9%	+8.0%
.788	**Number of Changes in Mouse Clicks Per Second in Past 15 Minutes**					
	>= 11 Changes	618	267	88.0%	69.8%	+1.9%
	< 11 Changes	65	56	12.0%	53.7%	-14.2%

Figure 6.5. The first seven features selected for the general model of office worker interruptibility shown in Figure 6.3.

The third feature selected is how long it has been since the active window was owned by NLNOTES.EXE, which corresponds to Lotus Notes, the integrated email and calendar software used by participants in this study. This seems to indicate that participants felt they were more interruptible when working with their email or calendar.

The fourth feature selected is also based on the output of the microphone-based silence detector. Whereas the first feature examines the output of the silence detector in the past second, the fourth feature is based on the maximum output in the past 5 minutes. Participants were more interruptible if there had been no loud noise at any point in the past 5 minutes. Like the first feature selected, this feature is probably capturing social engagement.

The fifth feature is how long it has been since the active window had a class name of length eleven (all windows have a title and a class, though the class is typically not visible to the user). Examining the full set of collected logs, 53.8% of windows with class name length of eleven are "SunAWTFrame" and 39.2% are "SWT_Window0". These window classes likely correspond to the use of the Eclipse development environment and related Java software. Participants were more interruptible if they recently shifted their attention away from these applications. In Chapters 8 and 9, we discuss the fact that this feature is not easily interpreted and comment on future directions for making features more easily interpretable.

The last two features are also related to computer usage. Participants were less interruptible if they had eleven or more top-level windows open at any point in the past 15 minutes. This might indicate that their attention was divided across many windows. The final feature is how many times the Mouse Clicks Per Second feature changed in the past 15 minutes. Eleven or more changes roughly corresponds to 6 mouse clicks, as each click typically results in 2 changes (from 0 clicks in second *t*, changing to 1 click in second *t + 1*, and then changing to 0 clicks in second *t + 2*). Participants were less interruptible when not generating mouse clicks in the past 15 minutes.

This model ignores the individual differences between the office workers, their responsibilities, and their work environments, but is still able to perform significantly better than the human observers from our previous study. This model captures social engagement using a microphone-based silence detector and a physical switch to detect when the office phone is off its hook. It captures task engagement based on the computer's active application and its set of open applications. While this chapter next examines differences in models that better describe the interruptibility of the office workers in this experiment, the fact that a single model can provide reliable results for all ten participants is an important result. For example, adaptive interfaces are often faced with the problem that they have no initial knowledge of the person to which they are adapting. While the software may ultimately be able to learn a person's preferences, this eventual utility is wasted if the person abandons the software due to initially poor performance. Our results here suggest that systems could be deployed with a default

	2 Manager Participants **Real Sensor Model**	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	141 64.4 %	8 3.7%
Highly Non-Interruptible	9 4.1%	61 27.9%
	Accuracy: 92.2% vs. 68.0% prior A': .960, z = 27.6, $p < .001$ vs. Human Observers, z = 11.7, $p < .001$	

Figure 6.6. The reliability of a naïve Bayes model of the interruptibility of our two manager participants.

initial model of office worker interruptibility. Because some conventions hold across a variety of office workers, this initial model could provide a more acceptable level of performance while the system is learning a person's individual preferences.

6.5 Models of Different Office Workers and Environments

It seems likely that differences in the responsibilities and working environments of our ten participants would manifest as differences in what features best predict their interruptibility. This section explores this question by developing separate models of the interruptibility of our manager participants, our researcher participants, and our intern participants. By focusing on office workers with similar responsibilities and work environments, we develop models that can more accurately identify "Highly Non-Interruptible" situations. We then examine differences in the features selected for models of different types of office workers.

6.5.1 A Model of Manager Interruptibility

Figure 6.6 presents the reliability of a model of the interruptibility of our two manager participants. This model is based on 20 features derived from the real sensors deployed in this study. With an accuracy of 92.2% and an A' of .960, this model performs significantly better than the 68.0% prior for this dataset (A' = .960, z = 27.6, $p < .001$) and significantly better than the human observers from our previous study (A' = .960 vs. A' = .720, z = 11.7, $p < .001$). It also performs significantly better than our general model of all ten office workers (A' = .960 vs. A' = .838, z = 5.93, $p < .001$).

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				68.0%	
.690	**Time Since Generating 9 Mouse Moves in a Second**					
	> 1 Minute	38	48	39.3%	44.2%	-23.8%
	>= 20 Seconds	21	11	14.6%	65.6%	-2.4%
	< 20 Seconds	90	11	46.1%	89.1%	+21.1%
.854	**Phone Off Hook at Time of Interruption**					
	Phone On Hook	144	32	80.4%	81.8%	+13.8%
	Phone Off Hook	5	37	19.2%	11.9%	-56.1%
	Undefined	0	1	0.5%	0.0%	-68.0%
.892	**Number of Changes in Key Press Per Second in Past 5 Minutes**					
	>= 30 Changes	84	19	47.0%	81.6%	+13.6%
	< 30 Changes	65	51	53.0%	56.0%	-12.0%
.910	**Number of Changes in Phone Off Hook in Past 15 Seconds**					
	< 2 Changes	107	42	68.0%	71.8%	+3.8%
	>= 2 Changes	42	28	32.0%	60.0%	-8.0%
.920	**Time Output of Silence Detector is Less Than 13 in Past 30 Seconds**					
	>= 12.5 Seconds (Quiet)	144	60	93.2%	70.6%	+2.6%
	< 12.5 Seconds (Noisy)	5	10	6.8%	33.3%	-34.7%
.926	**Using Mouse in the Past 15 Seconds**					
	Used Less than Half	118	70	85.8%	62.8%	-5.2%
	Used More than Half	31	0	14.2%	100%	+32.0%
.931	**Time Active Window Title is “New Memo – Lotus Notes” in Past 15 Minutes**					
	< 12.5 Minutes	123	66	86.3%	65.1%	-2.9%
	>= 12.5 Minutes	26	4	13.7%	86.7%	+18.7%

Figure 6.7. The first seven features selected for the model of manager interruptibility shown in Figure 6.6.

Figure 6.7 shows that the manager model of interruptibility is dominated by features that show the manager participants felt they were less interruptible when they were socially engaged. The first feature shows that the managers were less interruptible when they had not recently generated mouse move activity. In contrast, they were more interruptible when using the computer. As in other models discussed earlier in this dissertation, the second feature selected is whether the manager’s phone was off its hook at the time of an interruption. The third feature again captures an indication that the managers were more interruptible when using their computer, this time based on the number of seconds containing key press events in the previous 5 minutes. The fourth feature examines the noise level in the previous thirty seconds, with managers being less interruptible when their office was noisier. The fifth feature captures an indication that managers were more interruptible when they had generated mouse events for more than half of the previous 15 seconds (recall that we logged the number of mouse events in

	5 Researcher Participants Real Sensor Model	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	292 57.6 %	39 7.7%
Highly Non-Interruptible	45 8.9%	131 25.8%
	Accuracy: 83.4% vs. 65.3% prior A': .888, z = 22.6, *p* < .001 vs. Human Observers, z = 8.02, *p* < .001	

Figure 6.8. The reliability of a naïve Bayes model of the interruptibility of our five researcher participants.

each second, and this feature indicates that at least half of the previous 15 counts were greater than zero). The final feature shown here indicates that managers were more interruptible when they had spent most of the previous 15 minutes composing a new email.

6.5.2 A Model of Researcher Interruptibility

Figure 6.8 presents the reliability of a model of the five researcher participants. Based on 32 features, this model has an accuracy of 83.4% and an A' of .888. This is significantly better than the 65.3% chance performance for this data (A' = .888, z = 22.6, *p* < .001) and significantly better than the human observers from our previous study (A' = .888 vs. A' = .720, z = 8.02, *p* < .001). It is also significantly better than our general model of all ten office workers (A' = .888 vs. A' = .838, z = 2.39, *p* ≈ .017).

Figure 6.9 presents the first seven features selected for this model of researcher interruptibility. Consistent with the general model presented earlier in this chapter, the first two features selected are a threshold on the output of the microphone-based silence detector and whether the researcher's phone was off its hook at the time of the interruption. In this case, the selected feature is whether the silence detector has output zero for more than 4.5 of the past 30 seconds. The researchers are more interruptible under these quieter conditions.

The researchers were also less interruptible if the name of the active window's executable name was of length six in the previous 15 minutes. This is another example of a feature that is somewhat difficult to interpret, a point that Chapters 8 and 9 discuss as

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				65.3%	
.558	**Time Output of Silence Detector is 0 in Past 30 Seconds**					
	>= 4.5 Seconds (Quiet)	276	106	75.3%	72.3%	+7.0%
	< 4.5 Seconds (Noisy)	55	70	24.7%	44.0%	-21.3%
.650	**Phone Off Hook at Time of Interruption**					
	Phone On Hook	306	134	86.8%	69.5%	+4.2%
	Phone Off Hook	7	39	9.1%	15.2%	-50.1%
	Undefined	18	3	4.1%	85.7%	+20.4%
.718	**Time Active Executable Name Length is Equal to 6 in Past 15 Minutes (which primarily corresponds to “vs.exe” and “qw.exe”, the executable names for Microsoft Visual Studio and Quicken)**					
	Not at All	269	104	73.6%	72.1%	+6.8%
	< 2.5 Minutes	36	55	17.9%	39.6%	-25.7%
	> 2.5 Minutes	26	17	8.5%	60.5%	-4.8%
.778	**Day of Week**					
	Before Thursday	202	87	57.0%	69.9%	+4.6%
	Thursday or After	129	89	43.0%	59.2%	-6.1%
.799	**Distinct Values of Key Press Per Second in Past 15 Minutes**					
	>= 13 Distinct Values	196	78	54.0%	71.5%	+6.2%
	< 13 Distinct Values	135	98	46.0%	57.9%	-7.4%
.814	**Active Window Class Name Length Less Than 5 in Past 15 Minutes (which primarily corresponds to “tips”, a window shown by custom colleague availability awareness software use by participants)**					
	At Some Point	225	86	61.3%	72.3%	+7.0%
	Never	106	90	38.7%	54.1%	-11.2%
.824	**Minimum Number of Open Windows in Past Minute**					
	< 16 Windows	231	92	63.7%	71.5%	+6.2%
	>= 16 Windows	100	84	36.3%	54.3%	-11.0%

Figure 6.9. The first seven features selected for the model of researcher interruptibility shown in Figure 6.8.

an area for future work. But examining the active windows of the researcher participants, we note that 67.1% of active windows with executable name of length 6 correspond to “vs.exe”, which is the Microsoft Visual Studio integrated development environment. Another 32.7% correspond to “qw.exe”, which is Quicken personal finance software. Researchers were less interruptible if the active window had belonged to either of these software packages in the previous 15 minutes.

The fourth feature shows that researchers were less interruptible on Thursday and Friday, possibly due to a recurring meeting or a recurring deadline. While we did not collect participant calendar information, Horvitz and Apacible have demonstrated the utility of electronic calendars in their work on interruptibility models (Horvitz and Apacible 2003).

The fifth, sixth, and seventh features are related to researcher use of their computer. The fifth feature shows that the researchers were more interruptible when generating a variety of levels of keyboard activity in the previous 15 minutes. The sixth feature shows that they were also more interruptible if the active window class was of a length less than 5 at some point in the previous 15 minutes. For these researcher participants, 44.5% of such active windows correspond to “Tips” and another 9.8% to “Main”, both of which are associated with custom software used by these participants. This custom software showed sensed information about the availability of colleagues, including whether they were in their office, whether they were active on their computer, and whether they currently had an item scheduled on their calendar. This seems to indicate that our researcher participants were more interruptible if they had recently investigated the availability of a colleague. Another 44.5% of active windows with class name length less than 5 had a null class name. This appears to be due to a permissions issue, wherein our sensing software was denied access to the window class name. This occurred for windows owned by a variety of processes and we did not observe a systematic cause. Finally, the seventh feature shows they the researchers were less interruptible if they had 16 or more windows open at any point in the previous minute.

In contrast to the dominance of social engagement in our model of manager participant interruptibility, this model shows that researcher interruptibility was based on a mix of social engagement and task engagement. While the first two features in the manager model (more interruptible if using the mouse and less interruptible if on the phone) are sufficient to create a model with an A' of .854, the talking and phone features selected first by the researcher model only yield an A' of .650, significantly worse than the first two features in the manager model (A' = .854 vs. A' = .650, z = 5.07, $p < .001$). It takes 13 features for the researcher model to reach an A' of .854, and the full researcher model is significantly less reliable than the full manager model (A' = .888 vs. A' = .960, z = 3.01, $p < .01$). This seems to indicate that it was more difficult to model the interruptibility of the researcher subjects, potentially because it was difficult to capture their task engagement. The next chapter more carefully examines task engagement.

	3 Intern Participants **Real Sensor Model**	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	191 68.2%	12 4.3%
Highly Non-Interruptible	18 6.4%	59 21.1%
	Accuracy: 89.3% vs. 72.5% prior A': .903, z = 16.7, $p < .001$ vs. Human Observers, z = 6.79, $p < .001$	

Figure 6.10. The reliability of a naïve Bayes model of the interruptibility of our three intern participants in shared offices.

6.5.3 A Model of Intern Interruptibility in Shared Offices

As noted earlier in this chapter, we selected intern participants because our previous work indicated that a talking sensor is useful to models of human interruptibility and because the interns worked in an office with another intern. The interns were therefore regularly in the presence of a second person who might be engaged in a conversation or otherwise making noise, and we wanted to see affected a model of their interruptibility.

Figure 6.10 shows the reliability of a model of the three intern participants. Based on 38 features, this model has an accuracy of 89.3% and an A' of .903. This is significantly better than the 72.5% chance performance for this data (A' = .903, z = 16.7, $p < .001$) and significantly better than the human observers from our previous study (A' = .903 vs. A' = .720, z = 6.97, $p < .001$). It is also significantly better than our general model of all ten office workers (A' = .903 vs. A' = .838, z = 2.40, $p \approx .016$).

The first seven features of this intern interruptibility model are shown in Figure 6.11. The first feature selected is how often the silence detector has output zero in the past five minutes, with the interns being more interruptible in quieter offices. Note that this is a much longer time interval than is used for a similar feature in the researcher model (the researcher model examines only the previous 30 seconds). The second feature is how long it has been since the active window was owned by an executable with a name of length nine. Inspecting the data from the interns, 46.3% of such windows correspond to "javaw.exe" and another 37.0% correspond to "VCafe.EXE". For these subjects, these executables correspond to the use of the Eclipse integrated development environment and Symantec Visual Café, also a development environment. This feature indicates that the

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				72.5%	
.587	**Time Output of Silence Detector is 0 in Past 5 Minutes**					
	>= 2.33 Minutes (Quiet)	115	19	47.9%	85.8%	+13.3%
	< 2.33 Minutes (Noisy)	88	58	52.1%	60.3%	-12.2%
.677	**Time Since Active Executable Name Length is 9 (which primarily corresponds to "javaw.exe" and "VCafe.EXE", the executable names for Eclipse and Visual Café)**					
	> 15 Minutes	78	31	38.9%	71.6%	-0.9%
	>= 42 Seconds	55	10	23.2%	84.6%	+12.1%
	< 42 Seconds	70	36	37.9%	66.0%	-6.5%
.729	**Change in Active Executable Name Length Less Than 9 in Past 15 Minutes**					
	No Changes	121	65	66.4%	65.1%	-7.4%
	>= 1 Change	82	12	33.6%	87.2%	+14.7%
.765	**Time Since Output of Silence Detector is 15**					
	> 1 Second	162	68	82.1%	70.4%	-2.1%
	>= 0.1 Seconds	14	8	7.9%	63.6%	-8.9%
	< 0.1 Seconds	27	1	10.0%	96.4%	+23.9%
.785	**Time Door Cracked in Past 5 Minutes**					
	< 2.8 Minutes	176	76	90.0%	69.8%	-2.7%
	>= 2.8 Minutes	27	1	10.0%	96.4%	+23.9%
.800	**Change in Active Executable Name Contains "EXE" in Past 5 Seconds**					
	< 2 Changes	197	69	95.0%	74.1%	+1.6%
	>= 2 Changes	6	8	5.0%	42.9%	-29.6%
.814	**Day of Week**					
	Not Friday	180	76	91.4%	70.3%	-2.2%
	Friday	23	1	8.6%	95.8%	+23.3%

Figure 6.11. The first seven features selected for the model of intern interruptibility shown in Figure 6.10.

interns were more interruptible when they had recently changed the active window away from these development environments. The third feature appears to be related, counting how many times the intern changed to or from an active window with an application executable name of nine or more characters.

The fourth selection is a second silence detector feature, though this feature is somewhat unusual. Whereas every other silence detector used in a model has had the property that a noisier environment meant that subjects were less interruptible, the noisiest value of this feature (that the silence detector's output is currently 15) is associated with the interns being more interruptible. This may be because the interns were in a shared office, or it may be a timing issue caused by our experiment (because the timing of this feature is looking at the tenth of a second before a prompt, an undiscovered rounding error in the timestamp from our sensor logs could mean that the "noise" heard here is actually the sound of our audio file being played to request an interruptibility self-report), or this feature might just be a statistical anomaly.

The fifth feature shows that the interns were more interruptible when their doors were cracked for a significant portion of the previous five minutes (recall that our sensors detected whether a door was open, cracked, or closed). The sixth feature shows that an intern was less interruptible when there were 2 or more changes in whether the active window's executable name contained "EXE". This is most likely related to changing from "VCafe.EXE" to another application, then rapidly changing back. The final feature selected shows that interns were more interruptible on Fridays.

6.5.4 Discussion

This section has shown that more reliable models of human interruptibility can be created by focusing on office workers with similar responsibilities and work environments. Even though the general model presented in the last section performs better than the human observers from our previous study, each of the more targeted models in this section performed significantly better than the general model. Because of the differences in the responsibilities and work environments of the different types of participants, different sensors are better or worse indicators of their task engagement and social engagement.

For example, both the manager model and the researcher model gain much of their predictive power from features that capture indications of social interaction in a small time window preceding a prompt for an interruptibility self-report. Specifically, the researcher model starts with an analysis of the microphone-based silence detector in the previous 30 seconds. The second feature in both the manager and researcher models is whether the phone is off its hook at the time of the prompt for a self-report. In contrast, the additional noise in the shared offices of the interns means that the first feature selected for the intern model is based on the output of the microphone-based silence detector in the previous 5 minutes. Considering this longer time interval allows the model to be resistant to the noisier environment.

The differences in the features selected by the models, and the resulting improvement over a single general model, suggests that statistical models of human interruptibility should adapt to a person's individual preferences and circumstances. In Chapter 8, we discuss how this result informed our design of `Subtle`, specifically its support for continuous learning of individualized sensor-based models.

6.6 Models Based on Only a Laptop Computer

Given the results of our “Easy to Build” analysis in the previous chapter, we were interested in whether models of human interruptibility could be deployed based on only the sensing capabilities of a typical laptop computer. The results in the previous section, analyzing models of the interruptibility of different types of office workers, seem to support this possibility. While several models include the phone off hook physical sensor, only one includes a door sensor in its first seven features. We used a microphone installed in the office to detect talking, but many laptops include a built-in microphone that could potentially be used to detect nearby talking. The proximity of this microphone to the computer’s fan and keyboard could be problematic, however, as the sound of the computer’s fan or the person typing could interfere with the detection of nearby talking.

Of our ten participants, six primarily use laptops (participants 1, 2, 3, 5, 6, and 7). For these two managers and four researchers, we logged the output of our microphone-based silence detector using both the USB-attached microphone we installed in our sensing package and the built-in microphone of the laptop. While previous analyses considered the USB-attached microphone, this section considers models based on only the laptop’s built-in microphone and our software to log computer activity. While there are many corporate environments that could use software-based solutions to monitoring whether phones are in use, we decided not to include the phone sensor in these analyses because we want to examine the capabilities of a standalone laptop computer. We show that the sensing capabilities of a typical laptop computer, including its built-in microphone, can support a model of all six office workers with an accuracy of 79.3% and an A' of .836, significantly better than the human observers in our previous study (A' = .836 vs. A' = .720, z = 5.27, *p* < .001).

	6 Participants who use a Laptop Laptop Only Model	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	352 56.5%	59 9.5%
Highly Non-Interruptible	70 11.2%	142 22.8%
	Accuracy: 79.3% vs. 66.0% prior A': .836, z = 18.2, *p* < .001 vs. Human Observers, z = 5.27, *p* < .001	

Figure 6.12. The reliability of a laptop-based naïve Bayes model of the interruptibility of our six participants who use a laptop computer.

6.6.1 A General Laptop-Based Model of Interruptibility

Figure 6.12 shows the reliability of a model of our six participants who primarily used a laptop computer. Based only on the sensing capabilities of the laptop computer, this model is built from 48 features and has an accuracy of 79.3% and an A' of .836. This is significantly better than the 66.0% chance performance for this data (A' = .836, z = 18.2, *p* < .001) and significantly better than the human observers from our previous study (A' = .836 vs. A' = .720, z = 5.27, *p* < .001). It is not significantly different from the performance of our general model of all ten office workers based on our full sensor set (A' = .836 vs. A' = .838, z = 0.08, *p* ≈ .937).

The first feature selected for this model is a threshold on the amount of typing in the past 5 minutes, as captured by the number of changes in the output of our sensor which logged how many keys were pressed each second. The six laptop participants were more interruptible when they were typing more. The second feature is a threshold on the number of changes in the output of our silence detector in the previous 5 minutes. As our software outputs a value between 0 and 22, a large number of changes in this value indicates the presence of noise at a variety of volumes. Participants were less interruptible when the built-in microphone of the laptop computer detected this noise.

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				66.0%	
.568	**Number of Changes in Key Press Per Second in Past 5 Minutes**					
	< 53 Changes	209	161	59.4%	56.5%	-9.5%
	>= 53 Changes	202	51	40.6%	79.8%	+13.8%
.656	**Number of Changes in Output of Silence Detector in Past 5 Minutes**					
	< 364 Changes (Quiet)	206	62	43.0%	76.9%	+10.9%
	>= 364 Changes (Noisy)	205	150	57.0%	57.7%	-8.3%
.704	**Active Executable Name Length Less Than 6 in Past 15 Minutes (which primarily corresponds to "vs.exe" and "qw.exe", the executable names for Microsoft Visual Studio and Quicken)**					
	Never	348	139	78.2%	71.5%	+5.5%
	At Some Point	63	73	21.8%	46.3%	-19.7%
.733	**Size of Most Common Set of Open Windows in Past 15 Minutes**					
	< 17 Windows	319	123	70.9%	72.2%	+6.2%
	>= 17 Windows	92	89	29.1%	50.8%	-15.2%
.762	**Active Window Title Length Less Than 10 in Past 5 Minutes**					
	At Some Point	268	106	60.0%	71.7%	+5.7%
	Never	143	106	40.0%	57.4%	-8.6%
.772	**Maximum Mouse Move Per Second in Past 30 Seconds**					
	> 19 in Any Second	272	97	59.2%	73.7%	+7.7%
	<= 19 in Every Second	139	115	40.8%	54.7%	-11.3%
.781	**Day of Week**					
	Not Tuesday	311	183	79.3%	63.0%	-3.0%
	Tuesday	100	29	20.7%	77.5%	+11.5%

Figure 6.13. The first seven features selected for the laptop only model of six participants shown in Figure 6.12.

The third feature shows that these six laptop participants were less interruptible if the name of the active window's executable was less than 6 characters long at any point in the previous 15 minutes. Examining the sensor logs for these participants, 66.4% of such windows correspond to Microsoft Visual Studio, an integrated development environment, and another 32.4% correspond to Quicken, personal finance software. The fourth feature shows that participants were less interruptible if the most common set of open windows in the previous 15 minutes contained 17 or more windows. Subjects were also less interruptible if the active window's title was 10 or more characters long for the entire previous 5 minutes. Consistent with the first feature's showing that low levels of keyboard activity indicate a person is not interruptible, the sixth feature shows that low levels of mouse movement in the previous 30 seconds indicate participants were less interruptible. Finally, the last feature shown here indicates that participants were more interruptible on Tuesdays.

	2 Managers who use a Laptop **Laptop Only Model**	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	135 61.6%	14 6.4%
Highly Non-Interruptible	12 5.5%	58 26.5%
	Accuracy: 88.1% vs. 68.0% prior A': .940, z = 21.6, *p* < .001 vs. Human Observers, z = 9.37, *p* < .001	

Figure 6.14. The reliability of a laptop-based naïve Bayes model of the interruptibility of our two managers who use a laptop computer.

6.6.2 A Laptop-Based Model of Manager Interruptibility

Figure 6.14 presents the reliability of a laptop-based model of the interruptibility of our two manager participants. Built from 25 features, this model has an accuracy of 88.1% and an A' of .940. This is significantly better than the 68.0% chance performance for this data (A' = .940, z = 21.6, *p* < .001) and significantly better than the human observers from our previous study (A' = .940 vs. A' = .720, z = 9.37, *p* < .001). It is not significantly different from the performance of our model of these managers based on our full sensor set (A' = .940 vs. A' = .960, z = 0.78, *p* ≈ .438).

Figure 6.15 presents the first seven features selected for this laptop-based model of the interruptibility of our two manager participants. The first feature shows that the managers were more interruptible when using their mouse for 12 or more of the previous 30 seconds. The second feature shows that the managers were less interruptible when the laptop's built-in microphone detected nearby noise in the previous 5 seconds. Both of these features are consistent with our previous discussion that the manager interruptibility was dominated by social engagement and the managers were more interruptible when not engaged in a social interaction.

The third feature shows that the managers were less interruptible when they spend more than 5 of the past 15 minutes in an active window who's executable names was of length 12. Examining the data collected from the manager participants, 45.1% of such windows correspond to Microsoft Explorer's "explorer.exe", another 33.5% to Microsoft Internet Explorer's "iexplore.exe", and another 19.7% to Microsoft PowerPoint's "powerpnt.exe".

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				68.0%	
.633	**Time Mouse Move Per Second Equals 0 in Past 30 Seconds**					
	> 12 Seconds (Less Movement)	91	70	73.5%	56.5%	-11.5%
	<= 12 Seconds (More Movement)	58	0	26.5%	100%	+32.0%
.740	**Distinct Values of Output of Silence Detector in Past 5 Seconds**					
	< 5 (Quiet)	99	23	55.7%	81.1%	+13.1%
	>= 5 (Noisy)	49	47	43.8%	51.0%	-17.0%
	Undefined	1	0	0.5%	100%	+32.0%
.789	**Active Executable Name Length Equals 12 in Past 15 Minutes (which primarily corresponds to "explorer.exe", "iexplore.exe", and "powerpnt.exe", the executable names for Microsoft Explorer, Microsoft Internet Explorer, and Microsoft PowerPoint)**					
	< 5 Minutes	102	36	63.0%	73.9%	+5.9%
	>= 5 Minutes	47	34	37.0%	58.0%	-10.0%
.816	**Change in Key Press Per Second in Past Minute**					
	< 25 Changes (Less Typing)	69	119	85.8%	36.7%	-31.3%
	>= 25 Changes (More Typing)	30	1	14.2%	96.8%	+28.8%
.842	**Time Since There were Exactly 13 Open Windows**					
	>15 Minutes	105	50	70.8%	67.7%	-0.3%
	< 15 Minutes	6	10	7.3%	37.5%	-30.5%
	< 4 Minutes	38	10	21.9%	79.2%	+11.2%
.855	**Change in Active Window Title is "New Memo – Lotus Notes" in Past 5 Minutes**					
	< 6 Changes	132	70	92.2%	65.3%	-2.7%
	>= 6 Changes	17	0	7.8%	100%	+32.0%
.867	**Day of Week**					
	Not Thursday	123	66	86.3%	65.1%	-2.9%
	Thursday	26	4	13.7%	86.7%	+18.7%

Figure 6.15. The first seven features selected for the laptop only model of two managers shown in Figure 6.14.

The next three features are also related to computer usage. The fourth shows that managers were more interruptible when above a threshold on their amount of typing in the previous minute. The fifth feature indicates they were also more interruptible when exactly 13 windows were open at any point in the previous 4 minutes. The sixth shows that managers were more interruptible when they made several switches to or away from a window for composing a new email in the previous 5 minutes. This might be because they composed and sent several messages (switching back to the main email client window between each message), or they might have been switching to another

	4 Researchers who use a Laptop Laptop Only Model	
Self-Report	All Other Values	Highly Non-Interruptible
All Other Values	218 54.0%	44 10.9%
Highly Non-Interruptible	36 8.9%	106 26.2%
	Accuracy: 80.2% vs. 64.9% prior A': .860, z = 17.0, *p* < .001 vs. Human Observers, z = 5.74, *p* < .001	

Figure 6.16. The reliability of a laptop-based naïve Bayes model of the interruptibility of our four researchers who use a laptop computer.

application while composing a single message. Finally, the seventh feature indicates that the managers were more interruptible on Thursdays.

6.6.3 A Laptop-Based Model of Researcher Interruptibility

Figure 6.16 shows the reliability of a model of our four researcher participants who primarily used a laptop computer. Built from 26 laptop-based features, this model has an accuracy of 80.2% and an A' of .860. This is significantly better than the 64.9% chance performance for this data (A' = .860, z = 17.0, *p* < .001) and significantly better than the human observers from our previous study (A' = .860 vs. A' = .720, z = 5.74, *p* < .001). It is not significantly different from the performance of our model of all five researchers based on our full sensor set (A' = .860 vs. A' = .888, z = 1.04, *p* ≈ .298).

Figure 6.17 shows the first seven features selected for our laptop-based model of the interruptibility of the four researcher participants who primarily used a laptop computer. The first indicates that the researchers were less interruptible when the active window had recently been owned by an executable with a name of length six. In this data, 67.1% of such windows belong to Microsoft Visual Studio's "vs.exe" and another 32.7% to Quicken's "qw.exe". The second feature shows that subjects were more interruptible if the previous 5 minutes included a switch between an active window with a title of length 30 or greater and an active window with a title of length less than 30. These long titles seem to be commonly associated with programs that add a document title to the window title, including Microsoft Office applications, web browsers, and email clients.

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				64.9%	
.581	**Time Since Active Executable Name Length Equals 6 (which primarily corresponds to "vs.exe" and "qw.exe", the executable names for Microsoft Visual Studio and Quicken)**					
	> 15 Minutes	200	70	66.8%	74.1%	+9.2%
	< 15 Minutes	26	17	10.6%	60.5%	-4.4%
	< 2.5 Minutes	36	55	22.5%	39.6%	-25.3%
.681	**Change in Active Window Title Length >= 30 in Past 5 Minutes**					
	>= 1 Change	214	93	76.0%	69.7%	+4.8%
	No Changes	48	49	24.0%	49.5%	-15.4%
.724	**Time Output of Silence Detector < 6 in Past 15 Seconds**					
	>= 6 Seconds (Quiet)	183	83	65.8%	68.8%	+3.9%
	< 6 Seconds (Noisy)	79	59	34.2%	57.2%	-7.7%
.736	**Exactly 14 Windows Open for Previous Second**					
	False	221	134	87.9%	62.3%	-2.6%
	True	41	8	12.1%	83.7%	+18.8%
.753	**Maximum Output of Silence Detector in Past 30 Seconds**					
	< 22 (Quiet)	249	120	91.3%	67.5%	+2.6%
	>= 22 (Noisy)	12	21	8.2%	36.4%	-28.5%
	Undefined	1	1	0.5%	50.0%	-14.9%
.766	**Maximum Mouse Move Per Second in Past 15 Minutes**					
	>= 68 (More Movement)	166	71	58.7%	70.0%	+5.1%
	< 68 (Less Movement)	96	71	41.3%	57.5%	-7.4%
.783	**Change in Output of Silence Detector in Past Second**					
	< 5 Changes (Quiet)	220	106	80.7%	67.5%	+2.6%
	>= 5 Changes (Noisy)	42	36	19.3%	53.8%	-11.1%

Figure 6.17. The first seven features selected for the laptop only model of four researchers shown in Figure 6.16.

The third feature selected is a threshold on the last 15 seconds of output from our silence detector running on the laptop's built-in microphone. The researchers were more interruptible when the output of the speech detector was less than 6 for more than 6 of the past 15 seconds. The fifth and seventh features are also thresholds on the output of the silence detector, examining its maxima in the previous 30 seconds and the number of changes in its output in the previous second.

The fourth feature shows the researchers were less interruptible when they have exactly 14 windows open. That this feature is based on an exact count, instead of a threshold on the number of windows open, is another example of a feature that is difficult to interpret, a point we plan to discuss as an area for future work in Chapters 8 and 9. The sixth feature shows that researchers were more interruptible if they had generated 68 mouse move events in any second in the previous 15 minutes.

6.6.4 Discussion

This section has presented models based entirely on software-based sensing that can be deployed on a typical laptop computer. Using analyses of audio from a laptop's built-in microphone together with the desktop event stream, we have produced models that can identify situations reported as "Highly Non-Interruptible" significantly better than the human observers in our previous study. All of these models reach a level of performance that is not significantly different the previous section's models based on our full set of deployed sensors.

While the laptop's built-in microphone was the only mechanism available for these models to directly sense social engagement, the models also seem to have used the lack of computer activity as an indirect indication of social engagement. All three models presented in this section include at least one feature showing that greater levels of mouse or keyboard activity indicate that a person is more interruptible. If this greater level of activity were related to task engagement, we would expect it to indicate that the participants were less interruptible. Instead, it seems that these features capture the fact that people interact with their computers less when they are socially engaged. The features that are related to task engagement seem to be focused on what application a person is using, such as the researcher model feature that examines how long it has been since the active window was owned by Microsoft Visual Studio or Quicken.

6.7 Discussion

This chapter has shown that learned statistical models based on practical sensors can identify situations reported as "Highly Non-Interruptible" with a reliability that is significantly better than human observers. We have validated this finding with office workers who have a variety of responsibilities and working environments. Models focused on smaller sets of similar office workers perform better than a general model of all ten of our participants, even a general model performs significantly better than human observers.

It is particularly interesting to see that a typical laptop computer is sufficient for supporting reliable sensor-based statistical models of human interruptibility. While the full sensor models generally include the physical phone off hook sensor, the learner is able to compensate for the absence of this explicit indicator of social engagement by

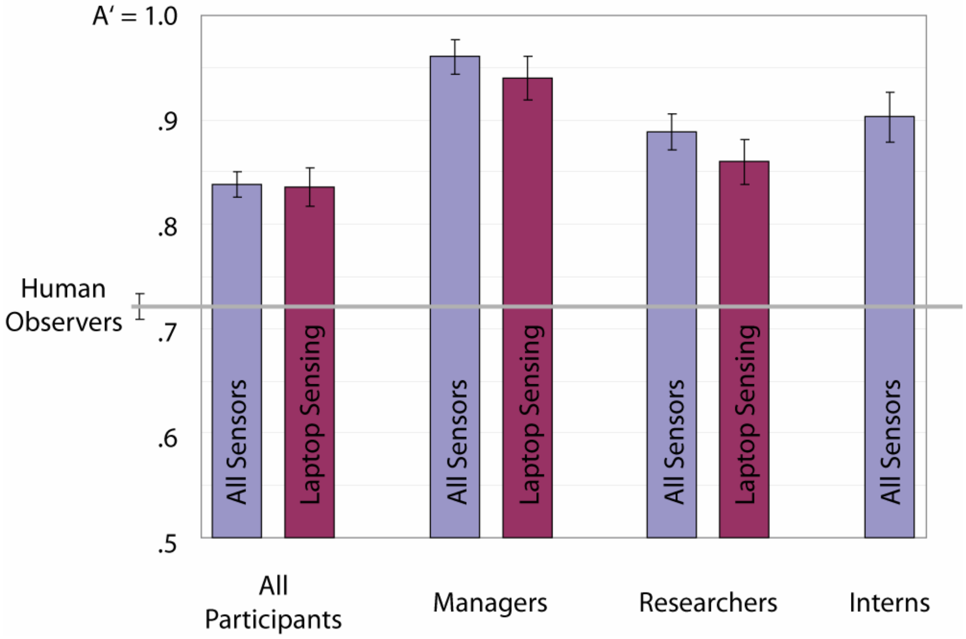


Figure 6.18. A summary of the reliability of the models in this chapter. All of our models identify "Highly Non-Interruptible" situations significantly better than human observers, and none of the laptop-only models are significantly worse than the corresponding full sensor model.

selecting microphone-based features and using the lack of computer activity as an indirect indication of social engagement. Figure 6.18 summarizes the reliability of models presented in this chapter. Every model performs significantly better than the human observers in our previous study and every laptop-based model reaches a level of reliability that is not significantly different from the corresponding model based on the full set of sensors.

The results presented in this chapter have several implications for the work we pursue in the remainder of this dissertation. First, this study installation included door sensors, motion detectors, a physical phone off hook sensor, and a special-purpose microphone placed away from ambient noise, but we also examined the sensing capabilities of a typical laptop computer. Given a laptop's built-in microphone, we demonstrated models that reach a level of reliability that is not significantly different from models based on our full sensor set. The remainder of this dissertation is therefore focused on sensing that can be deployed entirely in software on a typical laptop computer. While focusing on software-deployable sensing obviously limits the information available to a model, we are drawn to the fact that software-deployable systems can have real impact through wide availability. Second, the results in this chapter motivate a more careful examination of software-based indications of task engagement. We have seen that participants were often more interruptible when using their computer, but that activity in certain applications was an indication that they should not be interrupted. The next chapter more carefully explores models of interruptibility based on a computer's desktop event stream.

While this chapter has demonstrated the robustness of the results from our initial Wizard of Oz feasibility study and extended these results by examining the capabilities of a typical laptop computer, notable limitations remain. Because our data collection mechanism was installed in the office of each participant, we have studied the interruptibility of office workers only when they are physically in their office. It is possible that sensors we have shown to be predictive in an office may be less predictive outside of the office. This might be addressed by including location sensing in a model, but the exact impact of mobility on these types of models remains unknown. Various temporal issues may also exist, as we have examined interruptibility only at the moments in time for which we have collected self-reports. Considering these models in environments where they are continuously evaluated by an application might introduce additional concerns, such as whether a model's output is too discontinuous for use by a particular application.

Chapter 7

A Study of Task Engagement in a Natural Programming Problem

7.1 Introduction

Our results to this point have primarily been dominated by social engagement as it is captured by sensors to detect talking or phone use. While our models have included some features related to computer usage, these have been fairly high-level. Both our intuition and the literature suggest that capturing task engagement is important for creating reliable models of human interruptibility (Perlow 1999; Seshadri and Shapira 2001; Hudson *et al.* 2002). This chapter therefore seeks to more carefully examine task engagement in sensor-based statistical models of human interruptibility. Using our Wizard of Oz approach and transitioning to actual, implemented sensing, we examine how programmers respond to interruptions while they are programming and how statistical models can be used to predict their interruptibility. We develop a statistical model based on low-level input events in the development environment. This model can identify whether a programmer will choose to attend to an interruption with an accuracy of 75.6%, significantly better than the 58.5% prior (A' = .784, z = 18.5, *p* < .001).

Several motivations led us to investigate programmers. First, our results in the previous chapter suggest that the sensors we deployed could be significantly improved to better capture task engagement. Specifically, we expected the model of manager interruptibility to be dominated by social engagement and we expected that task engagement would play a larger role in the model of researcher interruptibility. The resulting model of the two manager participants is significantly more reliable than the model of the five researcher participants (A' = .888 vs. A' = .960, z = 3.01, $p < .01$). The fact that the researcher model is less reliable suggests that some difference between the managers and the researchers affected our ability to model their interruptibility, and it seems like this difference might be an inability of the deployed sensors to capture certain types of task engagement.

Beyond our desire to further investigate this aspect of our 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 result in programming errors (Anderson and Jeffries 1985; Ko and Myers 2004). 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 a computer, 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.

The next section introduces our experimental setup. We then discuss our exploratory data collection and the decision to use response time as a measure of programmer interruptibility. This is followed by a brief discussion of our Wizard of Oz analysis and our resulting implementation of an event sensing library. We then discuss our primary data collection using our implemented sensing. This is followed by a presentation of our resulting model and a discussion of its features.

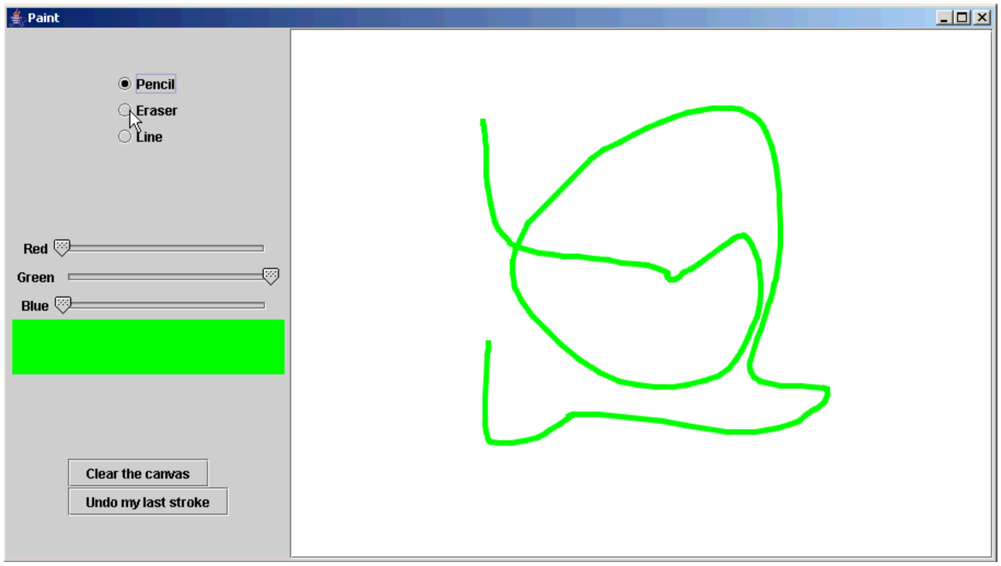


Figure 7.1. The *Paint* program that participants modified during the task.

7.2 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 recorded the entire screen at 12 frames per second in 24-bit color, with no noticeable impact on the computer's performance.

7.2.1 The Paint Program Primary Task

The *Paint* program, shown in Figure 7.1, 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 were:

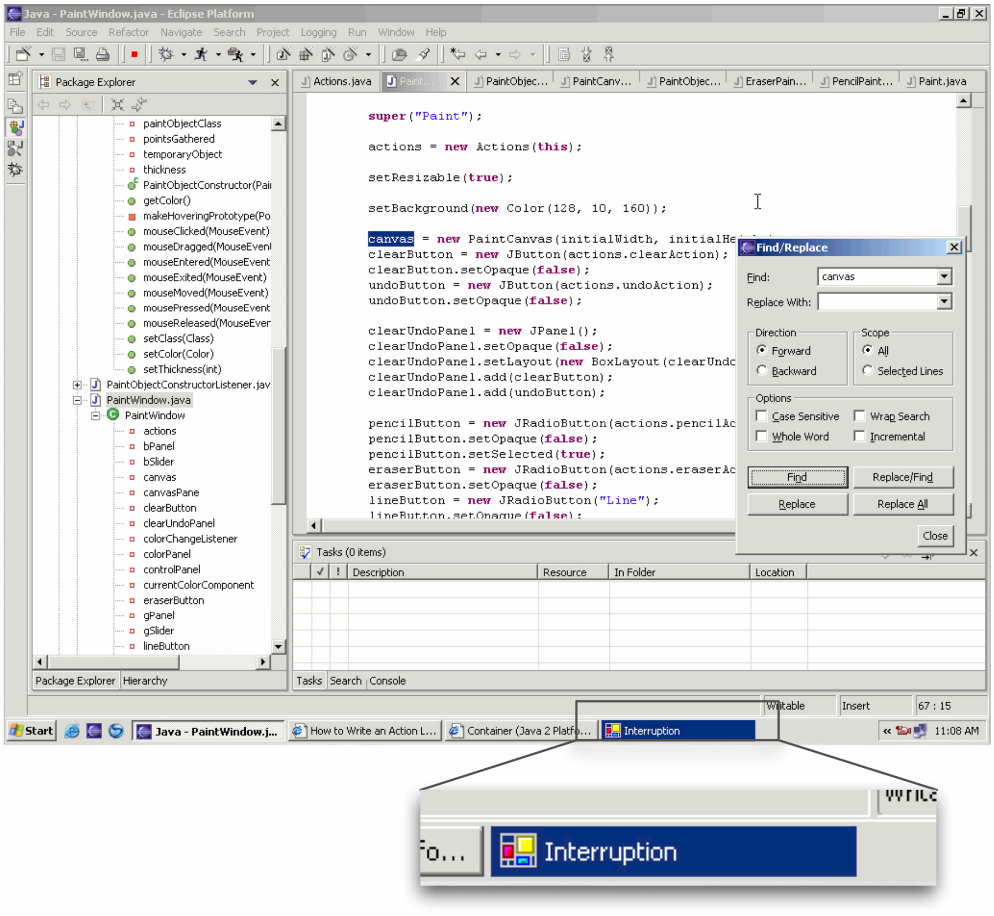
- "Users complained that scroll bars don't always appear after painting outside the canvas, but when they do appear, the canvas doesn't look right. Fix *Paint* so that (1) the scroll bars appear immediately when painting outside the visible canvas and (2) the canvas is correctly rendered when using the scroll bars to navigate the canvas.

- "Users complained that they can't selected 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 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 their strategy for ordering the requests and managing their time. They were told they would be paid $10 for each successfully completed request.

7.2.2 Mental Arithmetic Interruptions

At random time intervals averaging approximately once every three minutes, participants were notified of a pending interruption by an audio alert and the flashing taskbar icon shown in Figure 7.2. Participants could choose whether to address the interruption 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 (McFarlane 2002). 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 (Robertson *et al.* 2004). 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



```
super("Paint");

actions = new Actions(this);

setResizable(true);

setBackground(new Color(128, 10, 160));

canvas = new PaintCanvas(initialWidth, initialHe
clearButton = new JButton(actions.clearAction);
clearButton.setOpaque(false);
undoButton = new JButton(actions.undoAction);
undoButton.setOpaque(false);

clearUndoPanel = new JPanel();
clearUndoPanel.setOpaque(false);
clearUndoPanel.setLayout(new BoxLayout(clearUndc
clearUndoPanel.add(clearButton);
clearUndoPanel.add(undoButton);

pencilButton = new JRadioButton(actions.pencilAc
pencilButton.setOpaque(false);
pencilButton.setSelected(true);
eraserButton = new JRadioButton(actions.eraserAc
eraserButton.setOpaque(false);
lineButton = new JRadioButton("Line");
lineButton.setOpaque(false);
```

Find/Replace

Find: canvas

Replace With:

Direction: ● Forward ○ Backward

Scope: ● All ○ Selected Lines

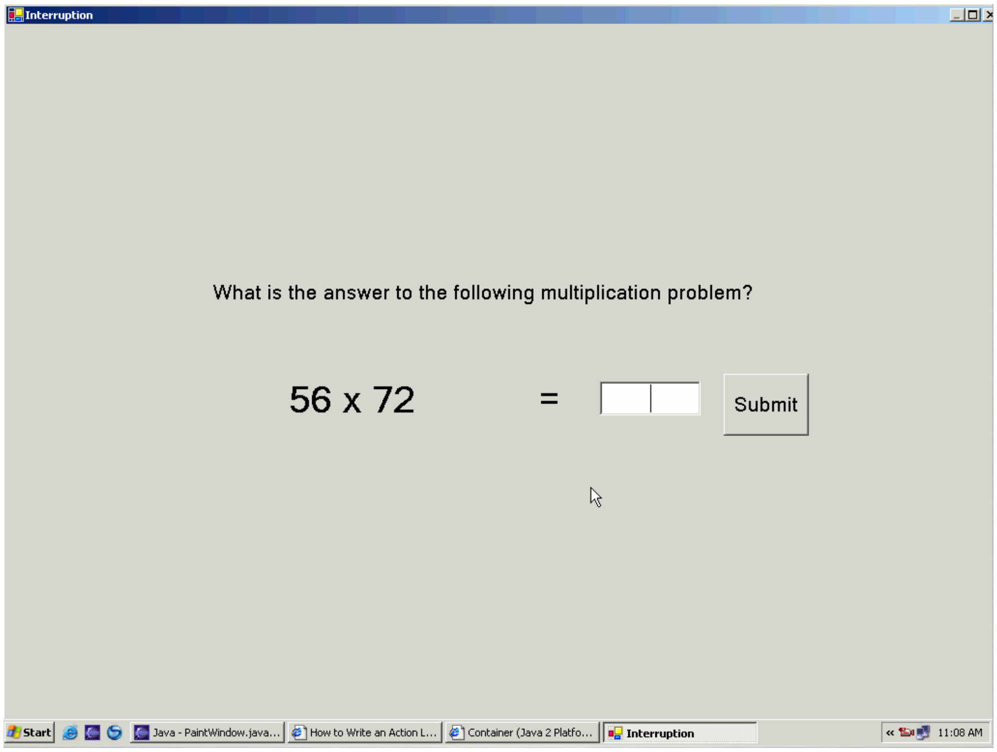
Options: □ Case Sensitive □ Wrap Search □ Whole Word □ Incremental

Find | Replace/Find | Replace | Replace All | Close

Figure 7.2. The Eclipse development environment,
with a pending interruption flashing on the taskbar.

participants that they would lose $2 for any ignored or incorrectly answered interruption, but we did not enforce this penalty.

When a participant clicked on a notification, a two digit multiplication problem was presented, as in Figure 7.3. 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 (Lemaire et al. 1996) and working memory failures are known to be a significant cause of programming errors (Anderson and Jeffries 1985; Ko and Myers 2004), 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 primary task during a 10 minute session of surfing web pages unrelated to programming.


Interruption
Submit
Start
Java - PaintWindow.java...
How to Write an Action L...
Container (Java 2 Platfo...
Interruption
11:08 AM

What is the answer to the following multiplication problem?

56 x 72 =

Figure 7.3. A mental arithmetic interruption.
Note that it obscures the entire screen.

7.3 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 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 informed the work presented in the remainder of this chapter.

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 interruptibility based in task performance, as opposed to more subjective measures like the self-reports used in our prior work or the retrospective labeling used by Horvitz and Apacible (Horvitz and Apacible 2003). 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 instances of an error being introduced as the result of an interruption. We believe this is because the negotiated coordination of interruptions, wherein participants chose when they wanted to attend to an interruption, allowed participants to carefully externalize their working memory into the development environment, as just discussed. We also note that Gillie and Broadbent found similar results when the environment contains the knowledge needed to resume an interrupted task (Gillie and Broadbent 1989). 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 development environments.

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

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

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

Task Switching	
ACTIVATE	Makes the Eclipse window active
OUTSIDE	Performs actions outside the Eclipse environment and the running *Paint* program
RUN	Executes the program with CTRL-F11, the Run button, or the menu item
TEST	Interacts with the running *Paint* 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 7.4. The 20 simulated sensors we used to examine our exploratory recordings of programmers working on the *Paint* task.

7.4 Wizard of Oz Sensor Exploration and Implementation

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 20 simulated sensors in 6 categories, shown in Figure 7.4. We chose these sensors because they occurred with reasonable frequency in the exploratory recordings, because they seem like they might relate to interruptibility, and because they might be reasonably 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 an 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, recency, and density of simulated sensor activation and built statistical models from these features, attempting to predict the interruptibility of the programmers. We do 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 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, recency, and density 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.

We implemented our sensing 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, so 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 and logs which methods of which classes are visible and how many lines of code from each of those methods is visible. This allows events to be associated with classes and methods, rather than just characters or screen coordinates.

7.5 Primary Data Collection

In order to examine the effectiveness of our implemented sensing, we recruited twenty additional participants. Eight were undergraduates majoring in computer science, six were undergraduates majoring in related fields, two had bachelor's degree 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 interruption response time observations from these participants.

In order to apply a classifier, we clustered participant response times using an Expectation Maximization algorithm, as implemented in the Weka machine learning toolkit (Dempster et al. 1977; Witten and Frank 1999). Given a set of observations 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 observations. 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 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*: $t(195) = 11.48$, $p < .001$).

	20 Programmer Participants **Development Environment Sensing Model**	
Self-Report	Interruptible	Engaged or Deeply Engaged
Interruptible	217 45.7%	61 12.8%
Engaged or Deeply Engaged	55 11.6%	142 29.9%
	Accuracy: 75.6% vs. 58.5% prior A': .784, z = 18.5, *p* < .001	

Figure 7.5. The reliability of a naïve Bayes model of the interruptibility of our twenty programmer participants.

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 considering two clusters, we obtained a cluster with a mean of 2.946 seconds with a standard deviation of 1.529 seconds and a cluster with a mean of 27.116 seconds and a standard deviation of 31.601 seconds. Comparing these clusters to the data, we felt that three clusters provided a more appropriate division of the data and found that moving to four clusters appeared to offer no improvement.

7.6 A Model of Programmer Interruptibility

After clustering programmer response times, we developed a classifier to distinguish *interruptible* observations from observations in the other two clusters, *engaged* and *deeply engaged*. As in the previous chapter, this model was developed using `Subtle`, which applies an iterative feature generation algorithm to automatically extract appropriate high-level features from the low-level event logs collected by our sensing plug-in. We again constructed a naïve Bayes classifier and evaluated it using a standard ten-fold cross-validation.

Figure 7.5 presents the reliability of the resulting model. Built from 26 automatically generated features, this model has an accuracy of 75.6% and an A' of .784, significantly better than the 58.5% prior for this dataset (A' = .784, z = 18.5, *p* < .001). If used to filter development environment notifications, this model could prevent 72.1% of notifications at inappropriate times while still allowing the immediate delivery of 78.1% of

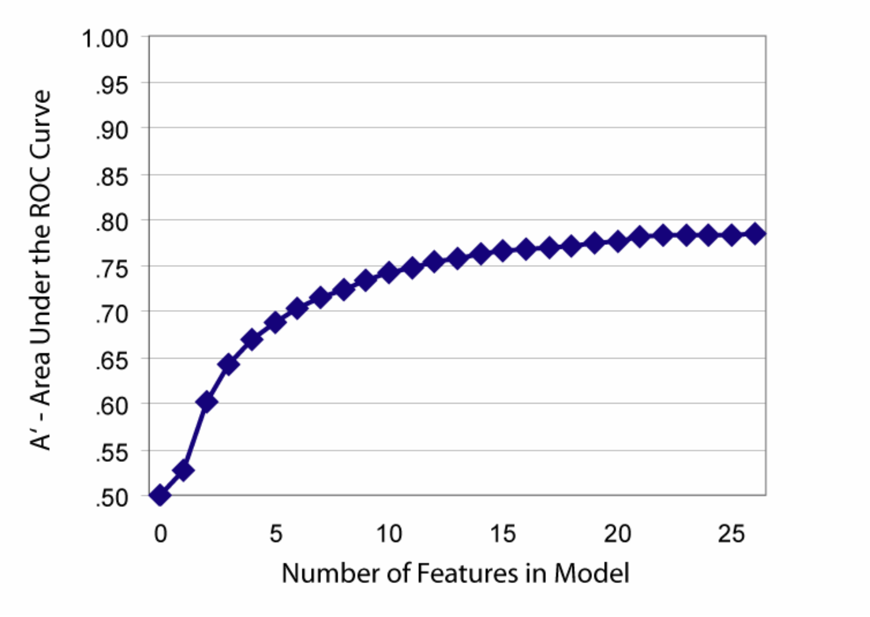


Figure 7.6. As with models presented earlier in this dissertation, the reliability of our model of programmer interruptibility jumps with its first features and grows more slowly as additional features are added.

appropriately-timed notifications. Figure 7.6 shows how this model's reliability is improved as features are added. The first several features again provide much of the model's eventual predictive power, though the dominance of the initial features is less pronounced than that shown in Figure 4.7 and Figure 6.4.

The first seven features in this low-level model of programmer interruptibility are shown in Figure 7.7. The first feature shows that programmers were less interruptible if they had generated an ExtendedModify event in the previous 15 seconds. This event is generated whenever the text of a file is modified. Interestingly, the programmers were less interruptible when this edit occurred inside the body of a method. This is consistent with the view that any editing includes a certain amount of working memory overhead, but that editing the definition of a method may require more focused attention than adding an import or variable declaration.

The second feature selected is that programmers were less interruptible if they had recently switched their focus away from the Eclipse integrated development environment. In the Standard Widget Toolkit (SWT) used to implement Eclipse, top-level windows are referred to as shells. Programmers were less interruptible if a shell deactivate event had

A'	Value	# Interruptible	# Highly Non Interruptible	% of Data	% Interruptible	% Change
.500	**No Features**				58.5%	
.527	**ExtendedModify Event in Past 15 Seconds**					
	No ExtendedModify Event	216	124	71.6%	63.5%	+5.0%
	Outside a Method	52	42	19.8%	55.3%	-3.2%
	Within a Method	10	31	8.6%	24.4%	-34.1%
.601	**Shell Event in Past Minute**					
	No Shell Event	174	133	64.6%	56.7%	-1.8%
	Shell Deactivate Event	55	52	22.5%	51.4%	-7.1%
	Other Shell Event	49	12	12.8%	80.3%	+21.8%
.643	**Mouse Move Event in Past Minute**					
	Move in Past 15 Seconds for > 37 Seconds of Past Minute	151	74	47.4%	67.1%	+8.6%
	Less Mouse Move Activity	127	123	52.6%	50.8%	-7.7%
.669	**Key Event Outside of a StyledText Widget in Past Minute**					
	No Key Events or Only in StyledText Widgets	234	188	88.8%	55.5%	-3.0%
	Key Event Outside of a StyledText Widget	44	9	11.2%	83.0%	+24.5%
.688	**Paint Event Outside of a Label Widget in Past Minute**					
	Paint Event Outside of a Label Widget	192	159	73.9%	54.7%	-3.8%
	No Paint Events or Only in Label Widgets	86	38	26.1%	69.4%	+10.9%
.703	**Any Tree Events in Past 30 Seconds**					
	No Tree Events	236	186	88.8%	55.9%	-2.6%
	1 or More Tree Events	42	11	11.2%	79.2%	+20.7%
.715	**Change in Mouse Events Within or Outside of Tree Widget in Past 15 Seconds**					
	No Mouse Events or < 4 Changes	259	173	90.9%	60.0%	+1.5%
	>= 4 Changes	19	24	9.1%	44.2%	-14.3%

Figure 7.7. The first seven features selected for the model of programmer interruptibility shown in Figure 7.5.

been generated in the previous minute, perhaps because the programmer had shifted to testing to *Paint* application or searching through documentation.

The third feature indicates that these programmers were more interruptible when generating mouse move events in the previous minute. This specific feature examines each second in a minute to see if a mouse move event occurred in the previous 15 seconds, applying a threshold on whether this is true for more than 37 seconds of the previous minute. Programmers might therefore generate a fairly continuous stream of mouse move events for at least 22 seconds, or a handful of mouse move event distributed across the previous minute. In either case, programmers were more interruptible.

The fourth feature is whether a programmer generated a Key event in a widget other than a StyledText widget in the previous minute. Because Eclipse uses StyledText widgets for editing code, this corresponds to typing in popup dialogs and other such widgets. Programmers were more interruptible when they recently typed in such a non-code widget.

The fifth feature shows that programmers were less interruptible when their actions resulted in the generation of Paint events in widgets other than labels. Many actions in the development environment generate these events, so this feature seems to indicate that general activity in the development environment meant that the programmers were less interruptible.

The sixth and seventh features are related to the tree view used to present the files, classes, and methods in an Eclipse project (see Figure 7.2). Programmers were more interruptible if they had generated any Tree events in the previous 30 seconds. This would seem to indicate that they were either exploring the list of methods in a class or moving from one source file to another. While they were more interruptible when generating Tree events, the programmers were less interruptible when moving between a Tree control and another widget. This is consistent with mousing over items in the tree view, as this generates a tooltip for the tree item (which the mouse is then over, changing what type of widget the mouse is moving over).

7.7 Discussion

This chapter has demonstrated learned statistical models that capture programmer task engagement in an integrated development environment based on the application's low-level event stream. Starting from a Wizard of Oz examination of screen capture recordings, we explored a variety of potentially predictive activities in the development environment. Because we found that editing was the best indicator of a high working memory load, we implemented an Eclipse plug-in to capture low-level events in the development environment. Using `Subtle`, we automatically extracted high-level features based on the frequency, recency, and density of low-level events in the development environment. The resulting model predicts whether programmers will choose to immediately attend to or defer an interruption with an accuracy of 75.6% and an A' of .784, significantly better than the 58.5% prior for this dataset.

The results presented in this chapter have informed how we address task engagement in the remainder of this dissertation. As discussed in the next chapter, `Subtle` addresses task engagement by examining the frequency, recency, and density of low-level events. An alternative approach would be to develop models of specific tasks and include routines that examine recent events to determine if a person is in the midst of a task, similar to the work by Bailey *et al.* (Bailey et al. 2005). While we ultimately believe that such approaches could be added to `Subtle`, our results in this chapter lead us to believe that we should initially focus on fully-automated approaches. We have shown that the iterative generation of high-level features based on the frequency, recency, and density of low-level events can capture indications of task engagement without the need for manually-specified models of tasks.

Several pieces of future work are suggested by the limitations of the work presented in this chapter. While we have demonstrated the automated development of models of task engagement within a particular application, it may be more difficult to develop comparable models across all of the applications used by an office worker. This is partially due to practical concerns, such as whether we are able to reliably access the underlying low-level event stream across applications, but it is also due to the large number of additional variables that are introduced when considering many additional applications. It is also unclear how the performance of the model presented in this chapter compares to other approaches or to human observers. Additional studies might examine how expert programmers would describe the interruptibility of a programmer at a given point in time. It might also be interesting to compare the performance of our model to other types of models, such as one that includes specific knowledge of common programming tasks. Finally, it is unclear whether the notion of task engagement studied in this chapter will provide a significant practical contribution to the models discussed in earlier chapters. While the notion of task engagement studied in this chapter is much richer than was considered in previous chapters, it might still be dominated by sensors related to social engagement. Further field studies are necessary to examine the utility of the types of task engagement models developed in this chapter.

Chapter 8

Toolkit Support for Sensor-Based Statistical Models of Human Situations

8.1 Introduction

This dissertation has shown that practical sensors can support reliable models of the interruptibility of office workers with diverse responsibilities in a variety of environments. We have shown that sensors capturing indications of social engagement and task engagement can support reliable sensor-based statistical models. These models can enable applications that consider human interruptibility and the social situation surrounding an application's usage.

This chapter presents `Subtle`, a tool we have developed to enable non-expert development and deployment of sensor-based statistical models of human situations. The design of `Subtle` has been informed by the results of our studies of models of human interruptibility, but the functionality provided by `Subtle` is sufficiently general that it can and should be applied to modeling a variety of human situations. For example, we have used `Subtle` to build an application that learns to automatically toggle whether a laptop's audio is muted, based on such factors as a person's location and what applications they are using. This application develops and uses a sensor-based statistical model with just sixteen lines of code. An instant messaging application could use `Subtle` to learn when to set or clear a busy flag shared with colleagues. A notification application could use `Subtle` to model whether a salient notification, such as a popup with an audio cue, or a peripheral notification, such as a fade-in system tray icon, is currently more appropriate.

This chapter starts by presenting the challenges addressed by `Subtle` and how our approach to these challenges has been informed by the work presented in this dissertation. We then present the details of `Subtle`'s implementation. We start by presenting an overview of `Subtle`'s architecture. We then introduce its extensible sensing library. This is followed by a discussion of `Subtle`'s use of type-based operators in a fully-automated iterative feature generation and selection algorithm. We then discuss `Subtle`'s support for continuous learning of individualized models on personal computers. Our discussion of implementation concludes with `Subtle`'s support for fields deployment, including the application of an extensible privacy policy to sensor logs. We then present a validation of `Subtle`'s support for developing applications that use sensor-based statistical models of human situations.

8.2 Challenges Addressed by Subtle

As a tool, `Subtle` seeks to enable the development and deployment of applications that use sensor-based statistical models of human situations. Motivated and informed by our work presented in this dissertation and related work, `Subtle` addresses four major obstacles to developing and deploying sensor-based statistical models.

8.2.1 Providing Relevant Sensors

Learning statistical models requires sensors which can provide relevant context. If the available sensors have little or no relationship to the concept being learned, no amount of processing can extracting a meaningful relationship. While this can ultimately be addressed only by an application designer, as it is the application designer who chooses what concept will be learned from what sensors, `Subtle` takes a two-pronged approach to helping the application designer address this problem. First, `Subtle` provides a library of sensors that we believe will be useful in many applications. Informed by prior work and by the studies presented in this dissertation, this library currently includes analyses of ambient audio, analyses of the desktop event stream, and WiFi-based location sensing. Second, `Subtle`'s sensor data collection and storage mechanisms are designed to support extensibility, easing the addition of new sensors.

8.2.2 Learning Appropriate Features

It is hard to know what aspects of available context are most useful in a statistical model. Non-experts can be very unsure about what types of features are appropriate for use with a machine learning algorithm, and even experts are unlikely to manually craft the best possible features (Markovitch and Rosenstein 2002). `Subtle` addresses this problem with a type-based iterative feature generation algorithm that applies operators to automatically examine a large number of potential features. Our current operators are based on the datasets collected and analyzed in this dissertation, and we believe they are appropriate for extracting a variety of meaningful high-level relationships from the types of sensors studied in this dissertation. `Subtle`'s extensible architecture also allows the addition of new operators that enable future sensors or different types of analyses. Because `Subtle`'s feature generation algorithm is fully automated, `Subtle` removes the need for an application designer to acquire significant specialized knowledge of machine learning techniques.

8.2.3 Accounting for Individual Differences and Unexpected Situations

It is hard for a static model to account for individual differences or unexpected situations. Our study of diverse office workers has shown that better models can be obtained by focusing on the particular environment and preferences of an individual. Outside the specific domain of interruptibility, it seems reasonable to expect important differences in how people expect an application to behave. If a model cannot adjust to individual preferences or differences in the environment, its utility in an application will suffer. `Subtle` addresses this problem by supporting continuous learning from data provided by individual users. Applications built with `Subtle` can therefore adapt to individual preferences and circumstances.

8.2.4 Managing Field Deployments

It is hard to manage the field deployment of applications that use sensor-based statistical models, thus limiting the scope and scale of research that can be conducted. Because sensor-based statistical models allow applications to adapt to the situations surrounding their usage, it is often important to study applications in real field deployments. But many practical issues arise in such deployments. Informed by the studies presented in this dissertation, `Subtle` focuses on software-deployable sensing for a typical laptop computer. This removes concerns over the costs of installing and maintaining custom hardware. `Subtle` also applies a privacy policy when storing sensor data, provides

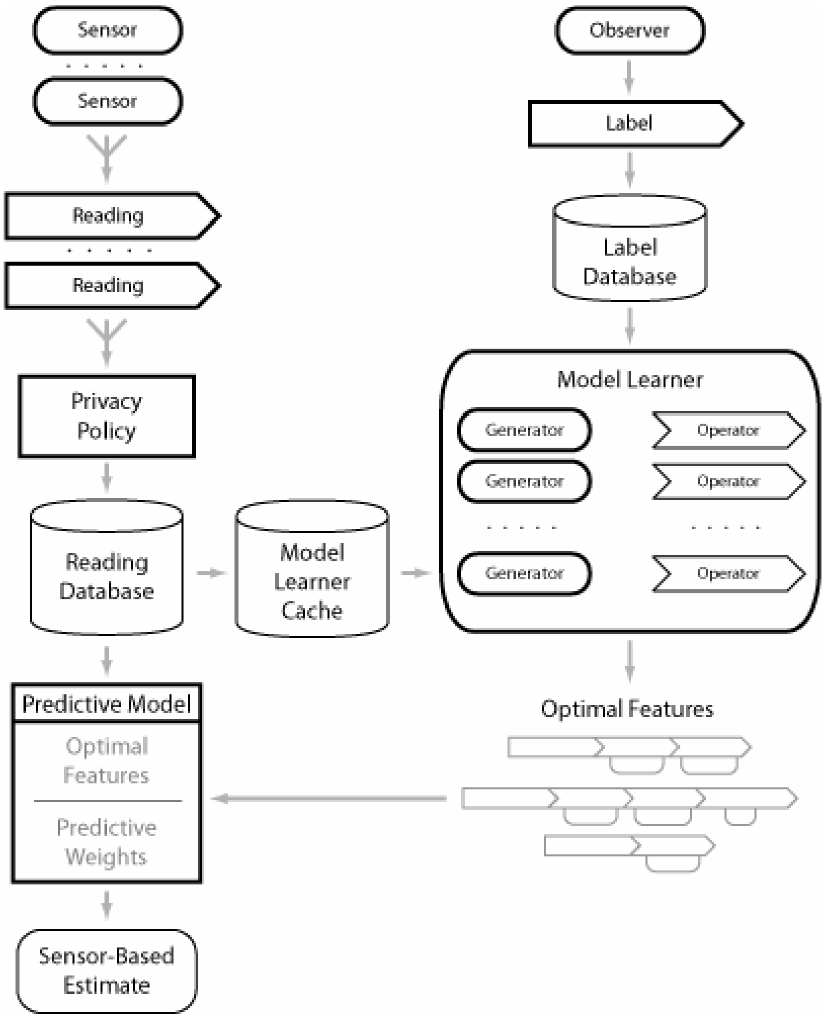


Figure 8.1. Overview of `Subtle`'s runtime architecture.

secure data and error collection, and supports the automatic application of signed code updates. Including this functionality in `Subtle` eases the development and deployment of applications that use sensor-based statistical models.

8.3 Subtle Architecture

Figure 8.1 presents an overview of `Subtle`'s architecture. We will discuss each component of this architecture as this chapter progresses, but here provide a high-level overview to serve as a basis for future sections. A variety of *sensors* provide input to `Subtle`, each generating *readings*. Each reading is either an *event* or a *state*, where events occur in an instant and states retain their value until either a new value arrives or a timeout occurs. Arriving readings are subjected to a *privacy policy* before being stored in a *reading database*. In parallel, *observers* occasionally collect *labels*, representing times at which the application knows the value of the concept being modeled. These labels are stored in a *label database*. At regular intervals, the *model learner* examines the reading database and the label database to produce a new model. The learner uses *generators* to iteratively consider *operators* that can be applied to the collected readings to build a set of high-level *features* appropriate for use in a statistical model. Once a model has been learned, it can be evaluated against the reading database to provide *estimates*.

The core of `Subtle` is implemented in Java with approximately 20,000 method lines of code. Several additional libraries are implemented in native code, including shared memory data structures used for inter-process communication and the sensing library discussed in the next section. `Subtle` runs in a separate process from the application that is using it, communicating with the application using XML-encoded procedure calls. We provide a Java client API that encapsulates this inter-process communication, but the underlying native library calls can be made from any programming language that can generate appropriate XML, invoke the native library call, and parse the resulting XML.

8.4 Extensible Sensing Library

Informed by the results of the studies presented in this dissertation, `Subtle` provides an initial sensing library that leverages the capabilities of a typical laptop computer. This library is implemented using standard Microsoft Windows libraries, so `Subtle` provides a proven set of sensors for all appropriate applications and hardware on the dominant platform. The sensing library currently includes analyses of ambient audio, analyses of the desktop event stream, and WiFi-based location sensing. We expect that this library will grow as additional sensors are developed for specific applications, as adding these sensors to our library will make them available to every application built with `Subtle`.

8.4.1 Ambient Audio Analyses

`Subtle` monitors the primary audio input device, typically a small microphone built into the case of a laptop computer. Because there are many different types of microphones with different characteristics, `Subtle` takes the approach of computing a variety of statistics from the microphone's audio stream. Any one of these statistics is probably sufficient for detecting nearby speech with a particular microphone in a particular environment, but we cannot know beforehand which feature will be useful in a particular model. Computing a variety of appropriate statistics gives our model learner the best opportunity to select appropriate features based on the ambient audio analyses.

Because it seems intuitive that the overall noise level could be a useful indication of how applications should behave in different environments, `Subtle` computes four features related to the overall noise level. `Subtle` analyzes both the audio signal's energy (defined here as the square of the magnitude of each sample) and power (defined here as the square of the difference between each pair of consecutive samples). Once per second, `Subtle` logs the average and the standard deviation of these values in the previous second.

Based on our previously presented findings that the presence of human speech is an important piece of context, `Subtle` computes several features intended specifically to identify speech. Analyzing the signal in the frequency domain, `Subtle` logs the energy level of audio in the range of human voice. Specifically, `Subtle` uses 1 KHz bins to log the energy of frequencies in the range of 0 to 4 KHz. `Subtle` also computes a set of features presented by Lu *et al.* (Lu *et al.* 2002). Specifically, `Subtle` logs several features based on the high-zero crossing rate ratio (HZCRR), the low short-time energy ratio (LSTER), and spectrum flux (SF). Lu *et al.* have shows that these features are effective for recognizing the difference between speech, music, and environmental sounds. These features have also proven effective in our prior deployment of a context-aware instant messaging client (Fogarty et al. 2004b).

8.4.2 Desktop Event Stream Analyses

`Subtle` logs a variety of events and states that have either been shown to be useful in our studies of human interruptibility or might otherwise be useful to applications built with `Subtle`. Because the active and open applications have proven useful, `Subtle` logs the window title, window class, and the executable name associated with each top-level window. `Subtle` also differentiates between top-level windows and popup windows (which are owned by another visible top-level window).

`Subtle` notes whenever a person opens, closes, moves, or sizes a window. Available information about the window, including whether it is a top-level window or a popup window, is logged for each of these events. `Subtle` also logs each mouse and keyboard input event, including available information about the widget in which an event occurred. Because our study of programmer task engagement showed that recent switches between windows or widgets can be a useful piece of context, `Subtle` logs each time a person changes the active window or changes the input focus within an active window.

Beyond examining open windows and input events, `Subtle` logs several additional pieces of information about computer usage and the status of the computer. For each active executable, `Subtle` logs the percentage of the CPU used in each five-second interval. While the semantic meaning of high CPU usage will vary across different applications, we believe this is a generally useful piece of context. For example, a media player application will likely exhibit higher CPU usage when playing content, as opposed to when paused or left open after a clip has finish playing. An integrated development environment can be expected to generate spikes in CPU usage when compiling. `Subtle` also logs whether the laptop's audio output is muted and whether the laptop is currently plugged in or running from batteries. Both of these seem to be an indication of a person's environment, as people may plug in their laptop when in a semi-permanent working environment or they may mute their laptop when in a public space.

8.4.3 WiFi-Based Location Sensing

While our field studies of models of human interruptibility have focused on office workers who are physically in their office, `Subtle`'s focus on sensing appropriate for a typical laptop computer raises location as an important piece of context. WiFi-based location sensing has been used in a number of projects, with the Place Lab initiative being one of the most successful examples (Schilit et al. 2003; LaMarca et al. 2005).

`Subtle` implements WiFi-based location sensing by using a laptop's wireless card to conduct periodic scans for nearby WiFi access points. `Subtle` logs the MAC address, network name, and signal strength of each detected access point. `Subtle` also logs to which access point the wireless card is currently connected. While some projects, including the Place Lab initiative, focus on translating WiFi MAC addresses into GPS coordinates, `Subtle` currently makes direct use of MAC addresses as identifiers for locations. For example, a person might be more interruptible when within range of access point *A* (which is near their office) and less interruptible when near access point *B* (which is near a conference room). Our discussion of operators and `Subtle`'s iterative feature generation process will include a discussion of how Subtle could be enhanced to use a Place Lab database to translate MAC addresses into GPS coordinates.

8.4.4 Sensing Library Extensibility Support

While the sensors provided by Subtle have been shown to be effective for modeling human interruptibility and we believe they will be effective in a variety of other applications, some applications will want to use custom sensing mechanisms. Subtle therefore supports two approaches to adding new sensing.

First, new sensors can publish XML-encoded readings into Subtle. This approach is appropriate when developing a new sensor for Subtle's sensing library or when an application has a piece of context that is probably not of general interest but that the application wants to store or provide to Subtle's model learner. When given appropriate XML, Subtle examines a reading as if it were generated by our provided sensor library: Subtle parses the XML to determine if the reading is a state or an event, filters it through the privacy policy, and stores the reading in the reading database.

A second approach to extensibility is appropriate when a large existing system needs to be integrated with Subtle. For example, consider if an application has already been built with the Context Toolkit (Dey et al. 2001), but the developer wants to add Subtle's machine learning support. It would be inappropriate to expect the developer to re-implement every sensor to generate XML in the form expected by Subtle. Instead, Subtle provides a degree of abstraction when querying the reading database. The default implementation stores reading values in our database using timestamped name/value pairs. But a developer can provide an arbitrary custom implementation for storing and recovering sensor readings. In the scenario described, an application developer could provide a custom implementation that queries the existing Context Toolkit database. Subtle could then learn from both the sensors in our library and the readings stored in the Context Toolkit database.

8.5 Fully-Automated Iterative Feature Generation and Selection

While the collection of appropriate sensor readings is necessary for learning a model, it is generally not sufficient. Models instead need to be built from features that extract higher-level concepts from the relatively low-level sensor readings. For example, the

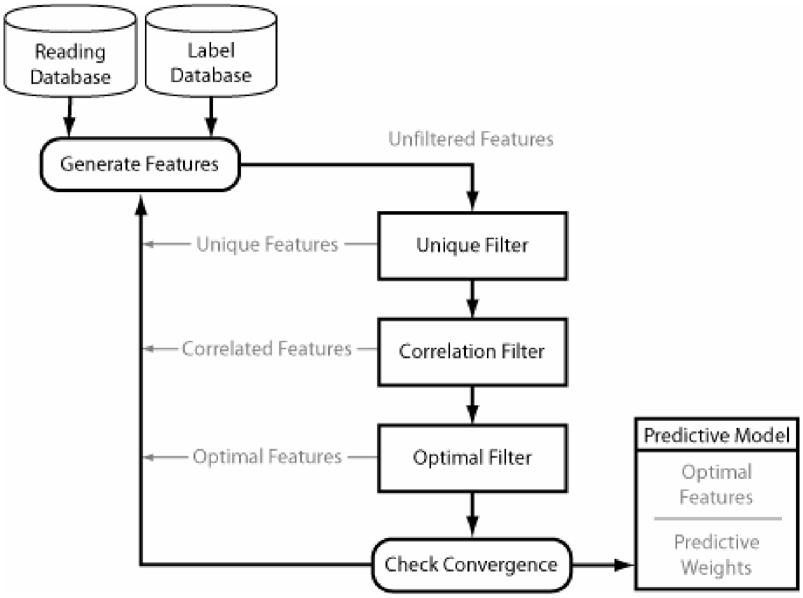


Figure 8.2. Overview of `Subtle`'s fully-automated iterative feature generation and selection algorithm.

exact value of a volume reading from an ambient audio sensor is probably less predictive than a Boolean feature computed by comparing the volume reading to a threshold. This feature might in turn be less predictive than one that captures whether the volume was above that threshold at any point in the previous 30 seconds. Creating appropriate high-level features is critical to successfully learning statistical models, but it is hard to know what aspects of available context are most useful in a model. Non-experts can be very unsure what types of features should be generated for use with a machine learning algorithm, and even experts are unlikely to manually craft the best possible features (Markovitch and Rosenstein 2002).

`Subtle` addresses this problem with a fully-automated iterative feature generation and selection process illustrated in Figure 8.2. Starting from the base set of features in the reading database, `Subtle` creates new potential features by applying *generators* to existing features. Each generator creates new potential features by adding an *operator* to the chain defining each existing feature. Because this process is fully automated, an application developer does not need to manually generate ideas for potential features or otherwise acquire specialized knowledge of machine learning techniques. Removing this need to acquire specialized knowledge eases the non-expert development of applications that use sensor-based statistical models.

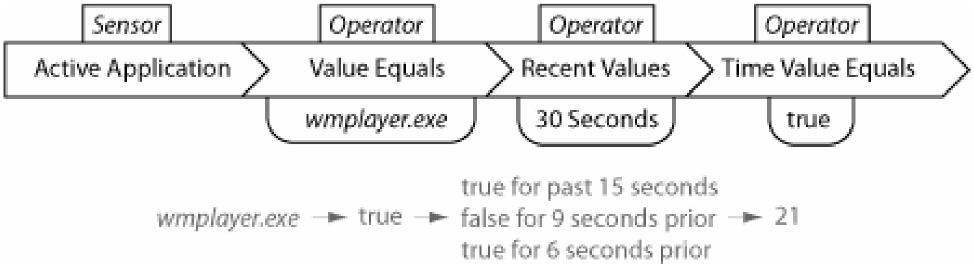


Figure 8.3. An automatically generated high-level feature to determine how many of the previous 30 seconds were spent in Windows Media Player.

8.5.1 Operator Chains

Figure 8.3 presents an example of a high-level feature constructed with an operator chain. Starting with a reading from an *Active Application* sensor, the first operator checks whether the value is equal to "wmplayer.exe". A second operator computed this value for the entire previous 30 seconds. Finally, a third operator determines how many of those previous 30 seconds had a value of `true`. These operators were added one at a time over the course of several iterations, and the resulting automatically-generated feature captures how many of the previous 30 seconds were spent in Windows Media Player. In an additional iteration, we might expect the creation of a feature that compares this value to a threshold, perhaps determining if a person spent more than 15 of the previous 30 seconds in Windows Media Player.

The parameters to each operator in Figure 8.3 were chosen by the generators that apply each operator. For example, the *Value Equals* generator examined the values of the *Active Application* sensor at the time of each label in the label database. Based on this history of values, the generator selected "wmplayer.exe" as value of interest to the *Value Equals* operator and created a new potential feature to examine it.

8.5.2 Potential Feature Filters

Because a very large number of potential features can be generated by the automated application of operators, `Subtle` applies a series of filters to potential features. The first filter removes non-unique features by finding sets of features that have equal values at the time of every label. Only one feature needs to be kept from each such set. In accordance with Occam's Razor (that the simplest explanation is preferred), `Subtle` selects the feature built from the fewest operators.

The second filter uses several computationally inexpensive heuristics to further reduce the number of features under consideration. It first examines the values of a feature at the time of each label. Features that have too many different values are filtered, as are features for which a value occurs in an overly small percent of the data. We filter these features to help prevent overfitting, a phenomenon where models mistakenly treat minor details of training data as important and therefore have an unnecessarily low accuracy when applied to new situations. The filter then uses several measures of correlation to select from the potential features. A machine learning researcher manually experimenting with a dataset could determine what notion of correlation is best for a particular dataset, but our fully-automated approach leads us to use several in parallel. The filter computes the information gain, gain ratio, and symmetrical uncertainty of each feature relative to the labels (Mitchell 1997; Yu and Liu 2003). The filter selects the *n* best-correlated potential features for each measure (features are selected if they are in the top *n* for any of the measures, where *n* was 1000 for all models presented in this dissertation). Finally, the filter uses Yu and Liu's notion of predominance to selected a small number features that are not in the top *n* but provide predictive power distinct from the top features (Yu and Liu 2003).

After applying the correlation filter, the number of potential features is reduced to a point where it is computationally appropriate to search for the optimal subset. Our final filter therefore uses wrapper-based selection to find an optimal feature subset (Kohavi and John 1997). Starting with an empty feature set, the algorithm adds and removes potential features until no change results in an improvement. The algorithm maintains a small set of the best results, enabling limited backtracking. The utility of each potential feature set is evaluated using a standard ten-fold cross-validation to estimate the area under the ROC curve (Hanley and McNeil 1982; Hand and Till 2001; Fogarty *et al.* 2005a). In this cross-validation, data is divided into ten folds and each fold is used to test a model trained from the other nine folds of data.

Given our earlier examination of different classifier algorithms and the fact that wrapper-based feature selection requires a computationally inexpensive classifier, `Subtle` currently chooses between naïve Bayes classifiers (Duda and Hart 1973; Langley and Sage 1994) and decision tree classifiers (Quinlan 1993). In the case where a decision tree is selected, the model type for each branch is recursively determined. This leads to either a naïve Bayes model or a tree with naïve Bayes models at its leaves. The

Iterations Completed	**Unfiltered**	**Unique**	**Correlate**	**Optimal**	**A'**	**(Acc)**
0	0	0	0	0	**.500**	.585
1	97	97	96	0	**.500**	. 585
2	3217	1322	212	13	**.633**	.636
3	19638	8247	1062	27	**.714**	.714
4	62894	23952	1192	42	**.751**	.733
5	81054	33227	1373	25	**.779**	.739
6	83061	34801	1383	27	**.787**	.760
7	83065	34805	1383	27	**.787**	.760
8	83065	34805	-	-	-	-

Figure 8.4. The number of features that pass each filter in each iteration of `Subtle`'s model learner. This example is from the learner for the programmer interruptibility model presented in Figure 7.5.

model type is determined during the wrapper-based selection of optimal features, based on which type of model yields a better score. This allows `Subtle` to adapt to the different types of data for which these different classifiers are more appropriate.

After the final filter determines the optimal subset of the current potential features, `Subtle` decides whether to continue by generating new potential features. Iteration terminates after a pass in which the generators do not create any new potential features or after five passes without any improvement in the scoring of the optimal features (using the estimated area under the ROC curve to evaluate progress).

Figure 8.4 illustrates `Subtle`'s feature generation and selection process by showing how many potential features clear the filters in each iteration of a model learning process. The counts shown are taken from the model learner session for the programmer interruptibility model shown in Figure 7.5. We have bolded the column for A', the area under the ROC curve, as a reminder that iteration continues until this column converges. The 97 events captured by our Eclipse plug-in are sufficiently low-level that they provide no initial predictive value. Applying our operators to these 97 potential features creates 3217 potential features, 1322 of which are unique. A second iteration applies operators to the 1322 unique potential features, creating 19638 potential features. As higher-level features are developed via the successive applications of operators, the model becomes more predictive. The fifth iteration, for example, results in a model that is more accurate and based on fewer features that the model resulting from the fourth iteration. This indicates that some feature generated by the application of an operator in the fifth

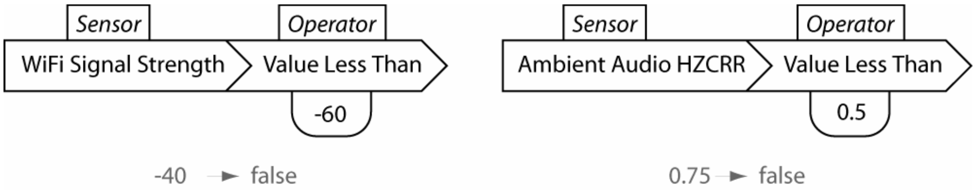


Figure 8.5. While the strength of a WiFi connection and the value of a statistic from an ambient audio analysis have very different meanings, both can have operators applied based on the fact that they are numeric values.

iteration was a large improvement over the features from previous iterations. While the number of potential features grows somewhat rapidly, the uniqueness and correlation filters quickly reduce the number of potential features under consideration in each iteration. The optimal filter then searches for the optimal subset of the potential features. In this case, model development converges after the eighth application of our generators does not produce any new unique features.

8.5.3 Type-Based Operators

`Subtle`'s operators are generally based on the *type* of a sensor reading, rather than the semantics of a particular sensor. For example, certain operations are appropriate with a numeric value, others are appropriate with a Boolean value, and still others are appropriate with a string. `Subtle`'s type-based approach can automatically adapt to the addition of new sensors, because generators examine the readings provided by sensors and apply appropriate operators. In contrast, an approach that defined what operators should be applied to a specific sensor would require a new explicit definition for every new sensor. Furthermore, the development of a new operator would then require an explicit definition of how that operator should be applied to every existing sensor.

Figures 8.5 and 8.6 provides two examples of how `Subtle`'s type-based operators can be applied to obtain high-level features from sensors with very different semantic meanings. In Figure 8.5, the *Value Less Than* operator is used to apply a threshold to both the signal strength of a WiFi access point and to the value of a statistic computed by our analysis of ambient audio. Though these two sensors have very different semantic meanings and require different thresholds, the generator examines the history of values output by each sensor and selects an appropriate threshold to create a new potential feature. Figure 8.6 shows longer operator chains demonstrating this same concept at a

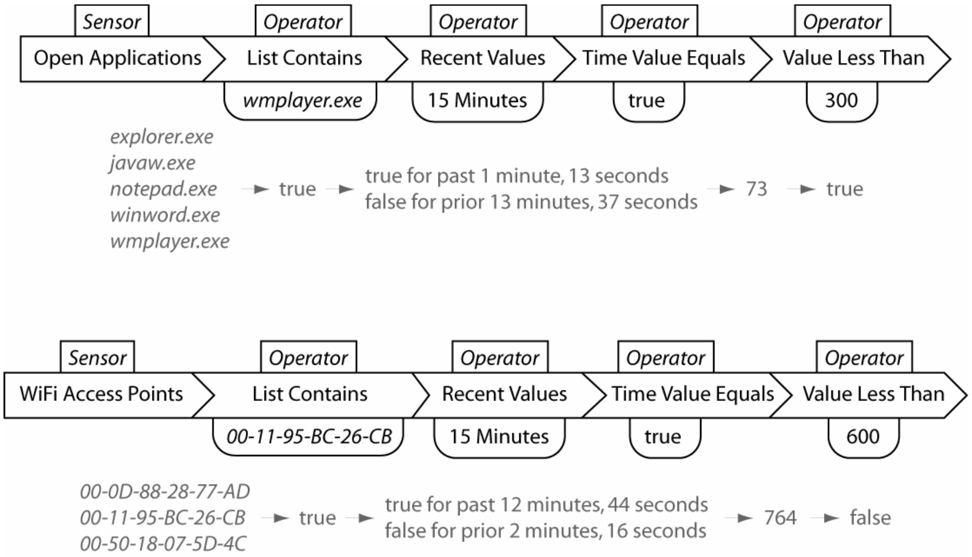


Figure 8.6. Applying the same type-based operators to low-level sensors to obtain high-level features with different meanings. The top feature captures whether Windows Media Player was open for more than 5 of the past 15 minutes, while the bottom captures whether a person was in range of a particular WiFi access point for 10 of the past 15 minutes.

higher level. Starting from the list of open applications, the first operator chain extracts whether Windows Media Player has been open for more than 5 of the past 15 minutes. The second operator chain starts from the list of WiFi access point and determines whether the person has been within range of a particular access point for more than 10 of the past 15 minutes.

`Subtle` currently defines types for Booleans, numeric values, strings, strings containing an XML document, a hash-based privacy-sensitive string type (introduced later in this chapter), and lists of values. New types can be defined by implementing an interface with a comparison method and methods for storing and retrieving a value from an XML stream. This section now presents the details of some of the operators we have implemented for `Subtle`'s current types. For the sake of brevity, we do not present an exhaustive operator list.

Numeric values are compared to thresholds by two different generators, *Discretize* and *Value Less Than*. Both use information gain to determine an information theoretic optimal threshold for a numeric feature. *Value Less Than* generates exactly one threshold for every existing numeric feature. *Discretize* uses Fayyad and Irani's method, creating multiple bins by recursively choosing thresholds according to information gain, with the minimum description length principle (MDLP) providing a stopping criterion (Fayyad and Irani 1993). Applying thresholds to numeric features allows those features to clear `Subtle`'s filtering of features with too many distinct values, and this type of discretization has been shown to lead to better results (Liu et al. 2002). We use both the *Value Less Than* approach and the *Discretize* approach because the MDLP criterion can be somewhat difficult to satisfy. *Value Less Than*'s generation of a single threshold allows every numeric feature an opportunity to emerge as predictive. But a single threshold is not always optimal, so *Discretize* allows multiple bins when the MDLP criterion indicates that the result is very likely to be predictive.

Using the comparison method required of all reading value types, `Subtle` provides several operators that can perform appropriate analyses of a variety of types. A *Value Equals* generator examines the value of an existing potential feature at the time of each label. It creates a new Boolean feature for each value that occurs often enough to warrant further examination. It is this generator, for example, that identifies “wmplayer.exe” as a potentially interesting value of the *Active Application* sensor in Figure 8.3. The *List Contains* generator performs a similar examination of features with a list type. In the example shown in Figure 8.6, the *List Contains* generator examined the values that appeared in the *WiFi Access Points* list at the time of each label and selected the given access point as warranting further examination.

Several history-based operators are also provided. For numeric values, `Subtle` computes such aggregates as the *Minimum*, *Maximum*, *Mean*, and *Median* of a value for some previous time interval. Based on the reading value comparison method, generic support is provided by a *Value Change Count* operator (how many times the value of a feature has changed in a time interval), by a *Most Common Value* operator (which value occurs most often in a time interval), by a *Time Since Defined* operator (how long it has been since a feature had a non-null value), and by a *Time Since Value Equals* operator (how long it has been since a feature had a particular value).

In creating `Subtle`'s operators and generators, our goal is not to provide a general learning algorithm (which would be the hard artificial intelligence problem). Instead, we have taken the lessons from our studies presented in this dissertation and created operators that are effective with the sensors provided by `Subtle`. `Subtle`'s learning mechanisms are appropriate for using sensed context to learn models of human situations and how application should behave, but it would be inappropriate to use `Subtle` for a problem like sketch recognition. We have also worked to ensure `Subtle` can be extended, both through our use of type-based operators and through some of the specific operators we provide.

8.5.4 Extensibility

While we have already discussed `Subtle`'s support for adding new sensors, developers can also add new operators to `Subtle`'s model learning process. This section discusses the extensibility of `Subtle`'s model learner using an example extension. Specifically, we discussion the integration of Place Lab's support for inferring GPS coordinates from WiFi MAC addresses (Schilit *et al.* 2003; LaMarca *et al.* 2005).

Including WiFi-based estimates of GPS coordinates in `Subtle` raises several issues. Laptops are often used indoors, and this may interfere with the reliability of available Place Lab databases. Specifically, Place Lab is trained through war driving and other mechanisms for collecting the GPS coordinates of WiFi access point, but GPS receivers generally do not work indoors. Place Lab databases are therefore unlikely to include the location of many indoor WiFi access points. It is for this reason, together with the fact that Place Lab databases are still very sparse in many areas, that `Subtle` is primarily focused on learning directly from the unique identifiers of WiFi access points. We note that `Subtle` can also already learn from a network's SSID. For example, all of the access points on the Carnegie Mellon campus have the same SSID. But `Subtle` currently cannot learn a concept like "somewhere in Pittsburgh" because `Subtle` has no way of knowing what WiFi access points are in Pittsburgh.

This capability could be added to `Subtle` through the definition of a new type and several additional operators. A type for a GPS coordinate could be defined as a pair of numeric values, storing the longitude and latitude of the coordinate. A *WiFi to GPS* operator could be created to check a set of WiFi MAC addresses against a Place Lab database, creating a new feature whose values are either a GPS coordinate or, in the case that the detected MAC addresses are not in the Place Lab database, a null estimate. A *GPS Distance* generator might then examine the history of GPS coordinates visited by a person, identify locations of interest using an algorithm like that presented by Ashbrook and Starner (Ashbrook and Starner 2003), and create new potential features based on the

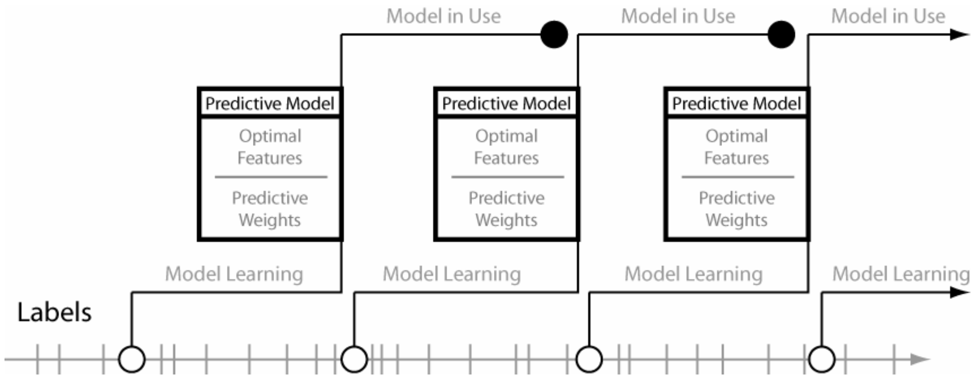


Figure 8.7. As additional labels are collected, `Subtle` continuously learns an updated sensor-based statistical model.

distance from a person's current location to a location of interest. `Subtle`'s existing operators would then be applied, allowing the automated exploration of a threshold on the distance from a location of interest or how long it has been since a person was in a location of interest.

Note that there is no need for a special operator to compute how long it has been since a person was near a location of interest. Once the *GPS Distance* operator computes a numeric value describing a person's distance from a location of interest, our existing *Value Less Than* operator can be applied to determine whether the person is nearby. After that, our existing *Time Since Value Equals* operator can consider how long it has been since the *Value Less Than* operator had a value of `true`. In this regard, `Subtle`'s type-based operators allow the developer of an extension to focus on operators that extract meaningful relationships in the domain of a new type (distance in this example). `Subtle`'s provided operators then explore a variety of manipulations of the value of the extracted relationship.

8.6 Continuous Learning of Individual Models

Because our studies found that better models can be created by focusing on office workers with similar responsibilities and work environments, we believe it is important that models adapt to an individual's environment and preferences. `Subtle` therefore includes support for continuous learning of personalized models from labels provided by an individual user.

Figure 8.7 illustrates `Subtle`'s continuous learning framework. The bottom timeline represents the occasional collection of labels by an application. When the model learner begins its iterative feature generation process, it takes a snapshot of available labels (represented by the vertical transition from the unshaded circle). The fully-automated process from the previous section is then executed against that set of labels. When complete, the resulting model begins to be used for making estimates (represented by the second vertical transition). More labels have probably been collected in the time taken to learn the model, so the process begins with a new snapshot. When this new model is finished, it is promoted into use. The lifetime of the original model ends at this point, marked by a shaded circle.

The computational cost of learning personalized models could be an obstacle to large deployments. When only a handful of users are involved in a small study, this can be solved by a dedicated server that collects and analyzes user data. But the cost of this approach grows with the number of people using an application. `Subtle` therefore executes the model learner on the client computer. It would be unacceptable for the model learner to continuously consume a large portion of a computer's processor, as this would both interfere with a user's other applications and quickly drain the battery of an unplugged laptop computer. `Subtle` therefore manages its own resource usage. When a person is active on their computer (indicated by mouse or keyboard activity in the previous 5 minutes) or when the computer is running from battery, `Subtle` limits the model learner to approximately 15 percent processor utilization (allowing 80 percent when both plugged in and idle). The model learner cache (shown in Figure 8.1) also reduces the computational cost of model learning by storing a variety of intermediate computations.

A common issue with learning personalized models is how an application should behave in the time before the first personalized model becomes available (including the time needed to collect labels and the time needed to learn the model). If a large set of labels have been collected from other people, they can be used to create a general model that is used until a personalized model becomes available. `Subtle` supports transitioning from a general model to a personalized model by exposing measures of the accuracy of each learned model. An application can use these measures to decide when enough labels have been collected to begin using a personalized model. An application might also choose not to use individual models, instead collecting labels to update one or more general models available to application users.

8.7 Supporting Field Deployments

In our experience conducting studies of human interruptibility and deploying a context-aware instant messaging client (Fogarty *et al.* 2004b), we have found a number of difficulties that arise in conducting studies or deploying applications based on sensor-based statistical models. A need for hardware installation and maintenance is an obvious difficulty. We address this in part by designing `Subtle` for a typical laptop computer. But `Subtle` also addresses the need to fix bugs discovered after deployment, the need to collect data during a field study, and the need to protect participant privacy in a field study.

Bugs will inevitably be found in an application after it is deployed, and widespread usage may reveal unexpected issues with a sensor implementation. `Subtle` therefore includes secure error collection and signed code updates. Uncaught exceptions are logged and the error files are uploaded to a central server. A deployment administrator can then examine these file for indications of a bug. If a code update is needed, `Subtle` supports signed code updates for both applications and the core `Subtle` process. Applications can be updated at startup, downloading new files from a central server. `Subtle` updates itself without need for an application restart (recall that `Subtle` runs in a separate process, so it can download updates and restart itself without a need for an application restart).

An administrator conducting a field study may need to collect detailed sensor logs from participants, and it can be very time-consuming to visit participants for the sole purpose of collecting these logs. Problems can also arise if a participant, perhaps unintentionally, deletes sensors logs. While it is hard enough when an administrator only needs to collect logs at the end of a deployment, it is common for an administrator to need to collect logs throughout a deployment. For example, our robustness field study continued until we had collected 100 self-reports from each participant. We therefore needed continuous data collection, so we could determine when the desired number self-reports had been collected. To support these types of requirements, `Subtle` provides for the secure, encrypted upload of collected sensor logs to a central server.

Learning a sensor-based statistical model requires the collection of detailed context, thus introducing concerns for participant privacy. Subtle therefore applies a privacy policy to every sensor reading before storing it in the reading database. An arbitrary policy can be implemented through extensions, but Subtle's default privacy policy is intended to provide appropriate protection without any effort from the application designer. The default policy is based in the belief that sensitive content is most likely to be revealed in window titles and other strings captured from the desktop (recall that Subtle does not record audio, another likely source of privacy issues). The default policy is therefore based on a privacy-preserving string type that uses a one-way cryptographic hash to mask the content of potentially sensitive strings. The default policy decides whether a string is potentially sensitive based on how the string was obtained, as opposed to being based on the content of a string. For example, the default policy considers window titles to be potentially sensitive (because they could reveal logins or other indicators of a person's identity), but does not consider the active executable filename to be potentially sensitive. It therefore tokenizes and applies a one-way cryptographic hash to window titles, but stores the actual value of the active executable name.

Subtle's default privacy policy is designed to preserve anonymity in collected sensor logs without any effort from an application designer. The default policy provides sufficient protection that our Institutional Review Board has approved a simple online consent form for use with Subtle, instead of the signed forms typically required in studies. But the application of a cryptographic hash can unnecessarily complicate exploratory data collection. Conversely, any cryptographic hash can provide strong protection only when it is hard to guess likely values. Because some applications will desire weaker or stronger privacy policies, Subtle supports the application specification of an arbitrary policy.

8.8 Subtle Validation

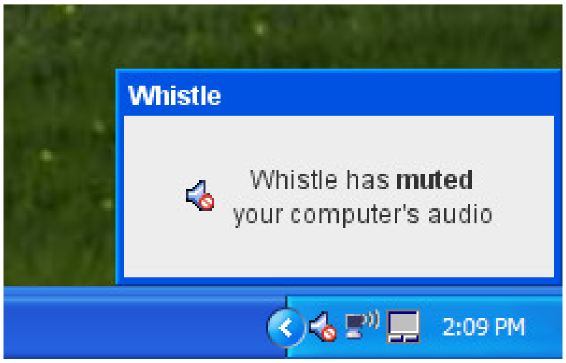
Evaluating a tool like Subtle is inherently difficult. While Subtle has been informed by a series of studies and Subtle enables applications that would previously have been extremely difficult to create, an evaluation based on a particular application or dataset provides only indirect insight into the utility of the tool (Edwards et al. 2003). We therefore seek to demonstrate Subtle's utility by examining two issues. First, as a demonstration of `Subtle`'s learning mechanisms, we note that all of the models presented in this dissertation were built using `Subtle`. Second, we consider two applications built with `Subtle`, discussing how `Subtle` reduces the time and effort needed to build applications that use sensor-based statistical models of human situations.

As a validation of `Subtle`'s model learner, we note that all of the models presented in this dissertation were built using `Subtle` (with the exception of the Wizard of Oz models, for which we loaded the original manually-generated features into `Subtle`'s feature selection process). While our prior publications included models based on features generated using manually-created scripts, we created the models in this dissertation by loading the unprocessed sensor data and labels into `Subtle`. In an indication that `Subtle` automatically generates features that are at least as useful as the features we manually crafted in our prior publications, all of the models presented here perform either better or not significantly different than the models from our prior publications.

While it might seem that it would be appropriate to conduct a validation by comparing `Subtle` to some other model learner, we note that the low-level events provided as input to `Subtle` are of little use in a classifier. This is evidenced in Figure 8.4, as the low-level events are of no use in a classifier until after `Subtle` begins to apply operators to extract higher-level features.

8.8.1 Whistle

Shown in Figure 8.8, *Whistle* is an application we have built with `Subtle`. Whistle addresses the problem that people often forget to mute or unmute the audio on their laptop. This can lead to disruptive email notifications in meetings or confusion when trying to play a media clip during a presentation from a muted laptop. Whistle monitors when people mute or unmute the audio of a laptop computer. It collects a label each time a person manually toggles their audio, learning a sensor-based statistical model that it uses to automatically mute or enable audio. This model is based on `Subtle`'s provided sensors, so it can learn such concepts as "*mute audio when I am away from my desk*" or "*enable audio when Windows Media Player is active*."



Whistle

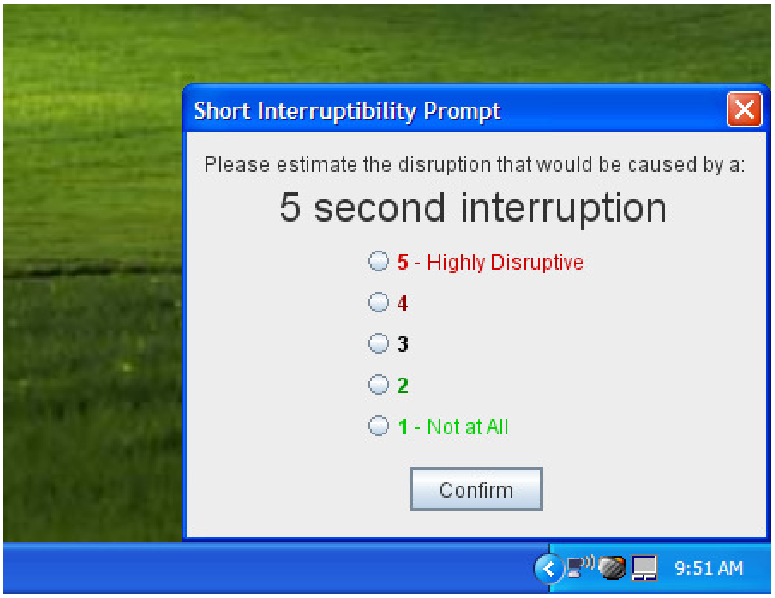
Whistle has **muted** your computer's audio

Figure 8.8. Whistle addresses the problem that people often forget to mute or unmute laptop computers. It collects a label each time a person manually toggles the mute flag, learning a model of the relationship between context and a person's desired setting.

Implemented in Java with a native library for querying and setting the operating system's audio mute flag, Whistle contains 265 lines of substantive code (including assignments and invocations while excluding imports, variable declarations, constant definitions, etc.). These lines of code are divided as follows:

- 110 lines are GUI-related, used to create a system tray icon, a popup menu for that icon, and the notification dialog shown in Figure 8.8.
- 100 lines are for querying and setting the audio mute flag. These lines are almost entirely within the native library.
- 16 lines are directly related to Whistle's use of `Subtle`. Of these, 6 are related to the overhead of starting and stopping the `Subtle` process and configuring Whistle's classloader to use the shared `Subtle` libraries. Another 4 are invoked every time a label is collected, connecting to `Subtle`, creating the label object, and providing the label to `Subtle`. The final 6 are invoked every time the model is evaluated, connecting to `Subtle`, obtaining the model, evaluating the model, and comparing the result to an action threshold.
- 27 lines are indirectly related to `Subtle`, as they coordinate timers that schedule when Whistle should evaluate the model and when it should check whether the person has manually toggled their audio.

As an example application, Whistle shows that `Subtle` greatly reduces the effort needed to implement an application that uses a sensor-based statistical model. Even using an



Short Interruptibility Prompt

Please estimate the disruption that would be caused by a:

5 second interruption

○ 5 - Highly Disruptive

○ 4

○ 3

○ 2

○ 1 - Not at All

Figure 8.9. AmIBusy Prompter learns individualized models of interruptibility, allowing applications to consider a person's interruptibility in just 6 lines of code.

existing machine learning toolkit like Weka (Witten and Frank 1999), it would not be possible to implement comparable feature selection in 16 lines of code, let alone feature generation, appropriate sensing, data storage, and privacy filtering.

8.8.2 AmIBusy Prompter

Figure 8.9 shows *AmIBusy Prompter*, another application we have built with `Subtle`. Based on our studies of human interruptibility, AmIBusy Prompter provides functionality similar to that of BusyBody (Horvitz et al. 2004). At configurable intervals, it displays a prompt like that shown in Figure 8.9. AmIBusy Prompter currently collects two types of self-reports: one asking about the disruption that would be caused by a 5-second interruption and one asking about the disruption that would be caused by a 15-minute interruption. The resulting interruptibility self-reports are provided to `Subtle`, which uses them to learn individualized models of a person's interruptibility. AmIBusy Prompter is implemented in approximately 250 lines of substantive code. As with Whistle, this code is overwhelmingly GUI-related.

AmIBusy Prompter makes the results of our work on human interruptibility available for inclusion in a variety of applications. Because AmIBusy Prompter is responsible for collecting labels and `Subtle` is responsible for learning a model, application developers can focus on how to best include a model of human interruptibility in an application. Only 6 lines of code are required to obtain an interruptibility estimate (2 lines of classloader configuration, 1 for connecting to the `Subtle` session created by AmIBusy Prompter, 1 for obtaining a model of interruptibility, 1 for evaluating the model, and 1 for comparing the result to a threshold). This makes research on how applications can best use models of human interruptibility accessible to the larger human-computer interaction research community, instead of just groups that include machine learning experts.

8.9 Discussion

This chapter has presented `Subtle`, a toolkit that enables the development and deployment of applications that use sensor-based statistical models of human situations. `Subtle` provides an extensible sensing library, fully-automated iterative feature generation and selection, continuous learning of personalized models, privacy filtering of collected sensor logs, and support for field deployments of applications. Informed by our studies of human interruptibility, `Subtle` removes many obstacles to using sensor-based statistical models in applications. `Subtle` thus moves the focus of human-computer interaction research onto applications and datasets, instead of the difficulties of collecting sensor data and learning statistical models.

`Subtle` is currently being used in research examining a variety of issues in human computer interaction and sensor-based statistical models. We will discuss our own plans in the Future Work section of the next chapter, but note that several other researchers are currently using `Subtle`. For example, interruptibility models developed by AmIBusy Prompter are being integrated into an instant messaging extension and a display intended for a person's office door. This work plans to examine how colleagues interpret the output of models of a person's interruptibility. `Subtle`'s sensing library has also been used in work on peripheral displays and to examine models of office worker productivity. The ease with which applications can directly access `Subtle`'s sensing library has also led to some surprising uses, including the use of `Subtle`'s sensing library to collect logs in a laboratory study of how people prioritize incoming email.

As a toolkit, `Subtle` aims to provide a low floor, meaning that it is relatively easy for non-experts to use `Subtle` to include sensor-based statistical models of human situations in applications. A common issue in toolkit design is the fact that a low floor often implies a low ceiling, referring to the point at which the abstractions and mechanisms provided by a toolkit begin to interfere with a developer's ability to implement an advanced functionality. In order to raise `Subtle`'s ceiling, many of `Subtle`'s most important components are extensible. But `Subtle` is not a general learning architecture, and `Subtle`'s focus is not on supporting the implementation of arbitrary learning algorithms.

Chapter 9

Conclusion and Future Work

9.1 Summary of Work and Specific Contributions

The promise of ubiquitous computing (Weiser 1991) and context-aware computing (Schmidt 2000; Dey *et al.* 2001) includes applications and devices that can sense and respond to an environment. Many contextual cues are non-ambiguous, as a sensor is either activated or it is not, but the ultimate decision about how an application or device should behave is often inherently ambiguous. For example, many email clients use an audio notification to announce the arrival of new email. Appropriate and useful when alone in a private office, this audio notification can be disruptive and socially awkward in an auditorium or during a meeting with colleagues. Current applications are generally unaware of the difference between these types of situations, and no single piece of context directly maps to "an audio notification is currently inappropriate." Given the vast number of potentially relevant pieces of context and the number of applications and devices that might want to consider this context, it is also unreasonable to expect a user to specify what action should be taken in every conceivable situation.

Sensor-based statistical models are one solution to this problem. Using a learned statistical model, an application can sense the environment, model a situation, and then act appropriately. This dissertation has examined sensor-based statistical models of human interruptibility, showing that models based on practical sensors can identify "Highly Non-Interruptible" situations significantly better than human observers. Using these types of models, application developers could create interfaces that negotiate entry into interruptions. This could reduce the damage caused by applications that demand attention at socially inappropriate times or allow workers to remain more focused on their tasks, while still allowing the benefits of appropriately-timed interruptions.

The work presented in this dissertation has been motivated by the goal of enabling sensor-based statistical models of human interruptibility. While it was initially unclear whether reliable models could be built, our Wizard of Oz feasibility study showed that models can be based on very practical sensors. Our later studies therefore focused on the robustness of practical sensor-based statistical models of human interruptibility and tools to enable their deployment in applications. Informed by the results of our studies, `Subtle` allows application developers to create effective models of human situations in as little as 16 lines of code. This section summarizes the contributions of this work, organizing our discussion according to the chapters in this dissertation.

9.1.1 A Wizard of Oz Approach to Top-Down Sensor Development

Chapter 3 presents a high-level discussion of our use of a Wizard of Oz technique to simulate potential sensors for in a sensor-based statistical model. This method is critical to the success of our work on human interruptibility. Without this method, it is very difficult to address the two requirements of a reliable sensor-based statistical model: that a sensor must reliably detect some piece of context and that the context must have a meaningful relationship to the concept being modeled. Later chapters apply our method to examining camera-based recordings collected in office environments and to examining screen-capture recordings in a study of programmer task engagement. The specific contributions of this work include:

- An explicit discussion of the intertwined nature of the two requirements of sensors in a sensor-based statistical model.
- The presentation of our Wizard of Oz method for addressing these requirements.
- A discussion of the benefits of our Wizard of Oz approach over a traditional bottom-up approach to sensor development. Our approach exposes both dimensions of a tradeoff between complexity and utility, enables iterative exploration of potential sensors, and allows the determination of which of the two requirements is not being met by a failed sensor.

9.1.2 A Wizard of Oz Study of Office Worker Interruptibility

Chapter 4 presents a study that collects audio and video recordings in the normal work environments of four office workers to consider a variety of potential sensors that might be useful in a model of their interruptibility. Examining simulated versions of sensors to detect 24 events or situations, we constructed a model that identified situations reported as “Highly Non-Interruptible” with an accuracy of 79.2% and an A' of .848. Because this model gains much of its predictive power from a handful of sensors and because several of those sensors are easily built, we examined models based on an “Easy to Build” set of sensors. Considering only a sensor to detect talking, phone use, mouse activity, keyboard activity, and time of day, we built a model with an accuracy of 78.6% and an A' of .797. The specific contributions of this work include:

- The finding that 32% of self-reports indicated that the office workers were “Highly Non-Interruptible”. This was the most common response on our five-point scale, suggesting that the office workers felt there were situations in which they were clearly non-interruptible. Their interruptibility in other situations may be more dependent on the nature of the interruption or some other factor.
- An examination of simulated versions of many sensors that seem likely to be useful in a model of human interruptibility.
- The finding that a sensor to detect nearby talking is a powerful indicator in a sensor-based statistical model of human interruptibility.
- The demonstration of a model based on an “Easy to Build” set of sensors. While the difference between the reliability of this model and the model based on the full sensor set is statistically significant, it is probably not large enough to warrant the costs of developing, installing, and maintaining the complex sensors used in the full model. This led us to focus on easy to build and low-cost approaches to sensor-based models of human interruptibility.

9.1.3 A Study of Human Observers Estimating Office Worker Interruptibility

Chapter 5 presents a study of human observers viewing the recordings from Chapter 4 and estimating the interruptibility of the office workers in the recordings. Viewing short snippets of the collected recordings, human observers found it difficult to estimate the interruptibility of office workers on a five-point scale. Examining the ability of the human observers to identify situations reported as “Highly Non-Interruptible” shows that they could do so with an accuracy of 76.9%. The specific contributions of this work include:

- The finding that human observers performed poorly when estimating office worker interruptibility on a five-point scale. This provides further support for our decision to focus on identifying "Highly Non-Interruptible" situations, as it is in these situations where people feel they are clearly not interruptible. Their interruptibility in other situations may be more dependent on the nature of the interruption or some other factor.
- The finding that human observers exhibited a bias towards estimating that the office workers were more interruptible than the office workers felt they were. This bias might be related to the self-interest of a person making an interruption.
- Showing that Chapter 4's models based on the full set of sensors and the "Easy to Build" set both identify "Highly Non-Interruptible" situations significantly better than human observers. That our "Easy to Build" sensors can support models that perform better than human observers provides further support for our decision to focus on practical and low-cost sensors.

9.1.4 A Robustness Study with Real Sensors and Diverse Office Workers

Chapter 6 presents a robustness study that implements actual sensors and then deploys them with a more diverse set of office workers than we included in our original Wizard of Oz study. We deployed sensors in the normal working environments of two managers whose interruptibility we expected to be dominated by social engagement, five researchers whose interruptibility we expected to include more task engagement, and three interns who worked in shared offices. While we obtained more reliable models when focusing models on participants with similar responsibilities and working environments, even a general model of all ten office workers still performed significantly better than the human observers in our previous study. Based on our previous results with our "Easy to Build" set of simulated sensors, we also examined models based on only the capabilities of a typical laptop computer. Using only a laptop's built-in microphone and the desktop event stream, we developed models that perform significantly better than the human observers from our previous study. The reliability of these models is also not significantly different from models based on our full set of implemented sensors. The specific contributions of this work include:

- Further support for our decision to focus on models that identify "Highly Non-Interruptible" situations. The distribution of self-reports obtained from the ten participants in this study is very similar to the distribution obtained from the four participants in our original Wizard of Oz study. "Highly Non-Interruptible" is again the most common response, accounting for 32.1% of self-reports.
- A demonstration of a model based on actual, implemented sensors that identifies "Highly Non-Interruptible" situations significantly better than human observers. The effectiveness of this model validates the results of our Wizard of Oz study, our implementation of appropriate sensors, and the robustness of our results with a variety of office workers.
- An examination of what features are most useful in models of the interruptibility of office workers with different responsibilities and working environments. For example, our models of managers and researchers examined audio features in the previous 30 seconds, but the noisier environment of the interns led the model learner to select an audio feature examining the previous 5 minutes.
- A demonstration of models of human interruptibility based on only the sensing capabilities of a typical laptop computer. Using only a laptop's built-in microphone and the desktop event stream, we demonstrated models that are significantly more reliable than human observers and not significantly different from models that also used the physical sensors deployed for this study.

9.1.5 A Study of Task Engagement in a Natural Programming Problem

Chapter 7 examines models of task engagement based on the low-level event stream in a programmer's integrated development environment. This work was motivated by differences in the reliability of the previous chapter's models of managers (dominated by social engagement) and researchers (whose responsibilities include more task engagement). In order to more carefully examine models of task engagement, we studied a total of 30 programmers completing a natural programming task while attending to occasional interruptions. Because we repeatedly observed programmers completing an edit or navigation before choosing to attend to a pending interruption, we measured their interruptibility as the time from the notification of a pending interruption until the programmer chose to attend to it. Using screen capture recordings from the first 10 participants, we simulated software-based sensors to detect a variety of activities in the development environment. Based on our analysis of these simulated sensors, we implemented an Eclipse plug-in to capture low-level events in the development environment. We then collected data from the final 20 programmers and used `Subtle` to automatically extract high-level features for a model of programmer interruptibility. The resulting model has an accuracy of 75.6% (versus a prior of 58.5%) and an A' of .784. The specific contributions of this work include:

- The examination of simulated versions of a variety of potentially predictive sensors in a programmer's development environment. Surprisingly, *EDIT* was the only simulated sensor that imposed enough of a working memory requirement and occurred often enough to emerge as predictive in our analyses.
- Development of an Eclipse plug-in to capture relevant events, including the classes and methods for events occurring within the code editing widget.
- Demonstration of a model built from high-level features automatically extracted from the captured low-level events. If used to filter development environment notifications at inappropriate times, this model could support the filtering of 72.1% of inappropriately timed notifications while still allowing the immediate delivery of 78.1% of appropriately timed notifications. This result indicates that indications of task engagement can be automatically extracted from low-level event streams, as opposed to the alternative view that a system must include manually-specified models of particular tasks.

9.1.6 Toolkit Support for Sensor-Based Statistical Models of Human Situations

Chapter 8 presents `Subtle`, our toolkit to enable the non-expert development and deployment of applications that use sensor-based statistical models of human situations. Informed by the sensors and modeling techniques proven effective in our work on sensor-based statistical models of human interruptibility and designed for a typical laptop computer, `Subtle` reduces the time and effort required to develop and deploy applications that use sensor-based statistical models of human situations. `Subtle` addresses the need for appropriate sensors by providing an extensible sensing library that currently includes ambient audio analyses, analyses of the desktop event stream, and WiFi-based location estimates. To address the problem that it is difficult to know what aspects of sensed context will be relevant to a statistical model, `Subtle` provides fully-automated iterative feature generation and selection. This is included within `Subtle`'s continuous learning framework, which allows `Subtle` to adapt to an individual user's preferences and environment. Because it is important to study sensor-based statistical models in realistic environments, `Subtle` provides several mechanisms to support application deployment, including the application of a privacy policy to collected sensor context, support for secure centralized error and data collection, and support for the application of signed code updates. Including this functionality in `Subtle` helps to enable realistic field deployments, allowing us to gain a better understanding of sensor-based statistical models in real applications. As a validation that `Subtle` lowers the barriers to developing applications that use sensor-based statistical models of human situations, we developed *Whistle* and *AmIBusy Prompter*. Whistle uses just 16 lines of code to learn a model of whether a laptop's audio should be muted. AmIBusy Prompter collects interruptibility self-reports and builds a model of a person's interruptibility, allowing applications to use just 6 lines of code to consider interruptibility. The specific contributions of this work include:

- An extensible and fully-automated iterative feature generation and selection algorithm appropriate for use by non-experts.
- Development of a system informed by the studies presented in this dissertation. `Subtle`'s choice of sensors and automated high-level feature development have been informed by and validated in our studies of human interruptibility.
- Making research on applications that use sensor-based statistical models accessible to the wider human-computer interaction research community, instead of only groups that include machine learning experts.

9.2 Future Work

The work presented in this dissertation suggests a variety of future work, including significant research that would be very difficult to pursue without a tool like `Subtle`. This section discusses four categories of future work. We start with a variety of studies that are already in progress or that we are interested in conducting with `Subtle`. We then discuss our interest in exploring models of human interruptibility based on implicit labels. The third area of future work we discuss is the exploration of how models can be based on data collected from many people. The final category of future work examines the fact that many features developed by `Subtle` are difficult to interpret, discussing possible approaches to developing human-interpretable features.

9.2.1 Conducting Studies with Subtle

Given the relative ease of developing applications and collecting field data with `Subtle`, there are a variety of applications we are interested in building and studies we are interested in conducting. AmIBusy Prompter currently collects two types of self-reports: one asking about the disruption that would be caused by a 5-second interruption and one asking about the disruption that would be caused by a 15-minute interruption. We are in the process of conducting a deployment to collect and model responses to these two different prompts. After verifying that people provide different responses to these prompts for an interruptibility self-report, we plan to examine differences in the features selected for a model of a short interruption versus features selected for a model of a long interruption.

We are also interested in how participants and their colleagues will interpret a sensor-based statistical model of interruptibility from this deployment. We therefore plan to deploy our learned models of interruptibility in the everyday lives of the participants. We are developing a door display and an instant messaging extension that will share estimates of a person's interruptibility with their colleagues, together with an indication of the sensed context that informed the estimate. After deploying these applications, we intend to meet with participants and their colleagues to discuss their reactions to and interpretations of the interruptibility models.

Many other applications can be built using `Subtle`. Whistle is an interesting example of an application because it uses a person's natural actions to learn a model of whether a laptop's audio should be muted. Collecting labels when people naturally choose to mute or unmute a laptop's audio, Whistle seems much more appropriate for real-world deployment than an application that introduces additional demands for explicit attention. We are also interested in developing an RSS client to investigate the possibilities for using sensor-based models when filtering or delaying notifications.

9.2.2 Using Implicit Labels in Models of Human Interruptibility

Applications like Whistle or an RSS client can learn a model of a very specific concept from implicit labels collected as people naturally interact with their computer, but we are also interested in how these very specific models might relate to a general model of a person's interruptibility. For example, consider that people may generally mute a laptop's audio when they are in a situation where it would be socially inappropriate for a laptop to play an audio notification. It seems reasonable to believe that whether or not a laptop is muted could therefore be a useful sensor in a model of a person's interruptibility (our studies did not examine this sensor, but we have included it in `Subtle`).

If whether or not a laptop is muted is useful in a model of human interruptibility, then whether or not a laptop *should* be muted may also be useful. An application like Whistle could observe a person's natural interaction with their computer to learn whether a laptop should be muted, and the output of Whistle's model could be useful to a more general model of interruptibility. Considering the implicit labels collected by Whistle might allow reliable models of human interruptibility to be based on fewer self-reports. It might even be possible to collect a variety of implicit labels from many different applications and build a general model of a person's interruptibility without ever explicitly prompting them.

As a first step towards examining this possibility, we intend to examine datasets collected from people using both Whistle and AmIBusy Prompter. It is too early to know what efforts will be fruitful, but it seems worthwhile to examine models of whether a laptop's audio should be muted to see if they share features with models of a person's interruptibility. It also seems worthwhile to build a model of whether a laptop's audio should be muted, then provide the output of that model as a feature to be used in developing a model of interruptibility.

9.2.3 Examining the Validity of Interruptibility Self-Reports

As noted in the conclusion of Chapter 4, this dissertation uses interruptibility self-reports as a ground truth measure of interruptibility but there are significant opportunities for research to examine the validity of interruptibility self-reports. The potential for this concern is motivated by significant evidence that self-reports can be influenced by a variety of factors. The actor-observer effect shows that people are more likely to attribute their own actions to situational factors and other people's actions to personal or dispositional factors (Jones and Nisbett 1971). This could manifest in interruptibility self-reports as a bias for participants to describe their interruptibility in terms of their environment. Because our models are based on the environment, this bias in participant self-reports could inflate the estimated reliability of our models. People also tend to be over-confident in assessing the reliability of social predictions (Dunning et al. 1990), so people might be expected to be overly confident in predicting how they would respond to an interruption. Shrauger and Osberg show that self-reports of psychological assessments are less reliable than judgments by other people, potentially due to a conflict with how a people perceive themselves (Shrauger and Osberg 1981). In a more humorous example of biases in how people perceive themselves, two separate studies have shown that over 90% of people surveyed report having an above average sense of humor (Allport 1961; Lefcourt and Martin 1986). There is a possibility that our interruptibility self-reports were influenced by how participants perceive themselves. If a participant feels they are a "busy" or "important" person, they might under-estimate their interruptibility. Conversely, a person who believes they are "accessible" might over-estimate their interruptibility.

One approach would be to examine the recordings collected in our initial Wizard of Oz feasibility work to determine how people respond to actual interruptions. If we examine the context at the time of natural interruptions, such as a phone call or the unexpected arrival of a guest, models based on self-reports should predict how the office worker will respond to these natural interruptions. There would, however, still be an issue of the difference between whether participants considered themselves interruptible versus whether they felt compelled to allow an interruption. Another approach might therefore be to collect several independent measures at the time of each self-report. For example, our study of task engagement might be modified to measure the time until a person chooses to attend to an interruption, collect a self-report of how the programmer perceives the severity of the interruption, and measure the time until it appears that the programmer has successfully resumed the interrupted task. Correlations between these different measures of interruptibility would provide evidence of their validity.

9.2.4 Building Models with Data Collected from Many People

The studies presented in this dissertation have produced better models when examining participants with similar responsibilities and working environments, but they have also shown that some features are useful across a wide variety of office workers. It seems important to explore methods for determining what types of features are generally useful, as opposed to being useful for a particular person or environment.

As we collect a variety of data from applications built with `Subtle`, we intend to explore what types of operators might lead to features that are generally useful. For example, a feature that indicates a programmer is less interruptible when using Microsoft

Visual Studio is extremely unlikely to be useful in a model of the interruptibility of a human resources employee. However, it may be the case that we can find generally useful features like "an office worker is less interruptible when typing in an application in which they have generated more than half of their keystrokes in the previous week." This type of feature might be able to capture task engagement without using the name of a specific application's executable and might therefore be more likely to transfer between office workers with different responsibilities.

Human-specified ontologies may also be useful in building models for data collected from many people. `Subtle` currently has no way to know that Microsoft Internet Explorer and Mozilla Firefox are similar applications. It also has no way to know that Lotus Notes and Microsoft Outlook both provide calendar and email functionality. Extending `Subtle` with ontology-related operators could allow the consideration of features based on the fact that a person is using a "web browser", as opposed to only considering whether they are using "iexplore.exe" or "firefox.exe".

9.2.5 Managing the Number of Potential Features Considered by Subtle

`Subtle`'s fully-automated feature generation process is based on the iterative application of operators. While `Subtle`'s current set of operators are designed to ensure an effective and efficient search of potential features, the extensible nature of `Subtle` introduces the possibility that a naïve extension could undermine `Subtle`'s model learner. For example, a naïve extension might perform some operation that combines every pair of existing potential features. The resulting rapid growth in the number of potential features would quickly reach the point where it would become computationally intractable to compute their values and examine their utility in a model.

We are interested in how `Subtle` can observe and manage the growth of the number of potential features considered in a model learning process. If the framework detects an operator generating an unacceptable number of new features, it could use a variety of approaches to filter the features generated by the problematic operator. A random approach may be sufficient, limiting each operator to generating *n* potential features at each stage in the iterative process and randomly selecting a subset of size *n* when the operator generates too many features. `Subtle` can also take advantage of the results of previous model learner results. If a particular feature was previously found to be useful, the filtering process should ensure that the feature is allowed the opportunity to emerge as predictive in the current model. Limiting the number of features considered would ensure that the model learner executes in a reasonable amount of time, while considering the results of previous model learner sessions should help to ensure that the learner is examining a useful subset of the potential features.

9.2.6 Developing Human-Interpretable High-Level Features

`Subtle`'s only consideration for the human interpretability of an automatically generated high-level feature is currently in the uniqueness filter, which chooses among features with the same value at every label by selecting the feature based on the fewest operators. Because it has no way to consider the fact that a very small gain in accuracy may lead to a feature that is very hard to interpret, we have seen that `Subtle` sometimes generates features that are very difficult to interpret. The *String Length* operator is a common culprit in the examples in this dissertation, as it allows `Subtle` to combine the data from two strings of the same length (such as "vs.exe" and "qw.exe") but it provides little insight into what the resulting feature actually means. As we begin to deploy these models in applications, this lack of interpretability will interfere with providing end-users with explanations of a model's behavior.

We are interested in exploring existing machine learning techniques that consider the cost of potential features. While cost is typically considered in either the financial or computational sense, we believe it is interesting to examine human interpretability as a cost when selecting features for a sensor-based statistical model. A simple notion of the cost of a potential feature could be obtained by counting the number of operators used to derive it, but the *String Length* operator is an example of an operator that seems to make interpretation more difficult than other operators. It might instead be better to assign variable costs to operators or combinations of operators. The specification of these costs could quickly become cumbersome, so it seems worth investigating how optimization techniques like those developed by Gajos and Weld might enable the specification of a large number of operator costs from relatively few examples (Gajos and Weld 2005).

Another approach to human-interpretable high-level features would be to explicitly model human-specified intermediate concepts. In the case of interruptibility, it seems clear that high-level concepts like "task engagement" and "social engagement" could be used as intermediate concepts. If hierarchical models were developed, an interruptibility estimate could then be explained in terms of the fact that a person appears to be socially engaged. This notion of social engagement could be related to a variety of sensors, including audio-based detection of a conversation, the presence of an item on a person's electronic calendar, or desktop events indicating that a person is engaged in an instant messaging conversation. Learning this intermediate concept therefore introduces the need to collect additional appropriate labels. An application like AmIBusy Prompter could do this by sometimes prompting for a person to describe their interruptibility, sometimes prompting for them to describe their level of "task engagement," and sometimes prompting for them to describe their level of "social engagement." From a tools perspective in the context of `Subtle`, using these human-specified intermediate concepts introduces the need to know what concepts are likely related to the labels provided by an application. For example, it may be inappropriate to use an intermediate concept like "task engagement" to explain why Whistle has decided to mute a laptop's audio. It is therefore interesting to consider whether a variety of generally appropriate intermediate concepts can be included in a tool like `Subtle`. Concepts that might be appropriate in a wide variety of applications include "in a conversation" (modeled from a variety of the audio features provided by `Subtle`), "away from office" (modeled from `Subtle`'s WiFi sensing), or "managing email" (modeled from the desktop event stream). Beyond being useful in explaining the estimates made by a model, the use of these intermediate concepts might also help to improve a model's reliability. Though models of intermediate concepts introduce the possibility of cascading errors, they also provide a point where significant knowledge can be accumulated (whether human-specified or automatically learned). For example, an explicit model of "in a conversation" is likely to be more robust than any single microphone-based feature.

9.3 Conclusion

Current applications and devices often stumble blindly through a human world they can neither sense nor understand. Because they are so obviously incapable, we blame ourselves for their shortcomings. For example, if a laptop interrupts a meeting to let everybody know that we have new email, we blame ourselves for not thinking to mute it. More importantly, the other people in the meeting also blame us. When we do remember to turn off our phone before attending a lecture, we may forget to turn it back when we leave. In that case, we are blamed by the person who is desperately trying to call. In either situation, the shortcomings of current applications and devices interfere with human needs and human interaction.

Sensor-based statistical models of human situations offer to change how we interact with computers. By examining models of human interruptibility, this dissertation has begun to show that practical, low-cost sensors can support models that capture important aspects of human situations and how people expect applications to behave. Including the results of this work in everyday applications could provide us with computers that better understand and adapt to the human situations surrounding their use. Comparing our models of interruptibility to estimates made by human observers suggests that computers might be reasonably polite partners in an interaction. We might then be able to spend less time dealing with technology and more time focused on our actual goals.

Bibliography

Abowd, G. and Mynatt, E. D. (2000). Charting Past, Present, and Future Research in Ubiquitous Computing. *ACM Transactions on Computer-Human Interaction (TOCHI)* **7**(1). 29-58.

Adamczyk, P. D. and Bailey, B. P. (2004). If Not Now, When? The Effects of Interruption at Different Moments Within Task Execution. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*. 271-278.

Allport, G. W. (1961). *Pattern and Growth in Personality*. New York: Holr, Reinhart, and Winston.

Anderson, J. R. and Jeffries, R. (1985). Novice LISP Errors: Undetected Losses of Information from Working Memory. *Human-Computer Interaction* **1**(2). 107-131.

Ashbrook, D. and Starner, T. (2003). Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. *Personal and Ubiquitous Computing* **7**(5). 275-286.

Avrahami, D., Gergle, D., Hudson, S. E. and Kiesler, S. (2006). Improving the Match Between Callers and Receivers: A Study on the Effect of Contextual Information on Cell Phone Interruptions. *Behavior and Information Technology (BIT)*. To Appear.

Avrahami, D. and Hudson, S. E. (2004). QnA: Augmenting an Instant Messaging Client to Balance User Responsiveness and Performance. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*. 515-518.

Avrahami, D. and Hudson, S. E. (2006). Responsiveness in Instant Messaging: Predictive Models Supporting Inter-Personal Communication. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2006)*. To Appear.

Bailey, B. P., Adamczyk, P. D., Chang, T. Y. and Chilson, N. A. (2005). A Framework for Specifying and Monitoring User Tasks. *In Press, Computers in Human Behavior, Special Issue on Attention Aware Systems*.

Bao, L. and Intille, S. S. (2004). Activity Recognition from User-Annotated Acceleration Data. *Proceedings of the International Conference on Pervasive Computing (Pervasive 2004)*. 1-17.

Barker, R. G. (1968). *Ecological Psychology*: Stanford University Press.

Begole, J. B., Matsakis, N. E. and Tang, J. C. (2004). Lilsys: Sensing Unavailability. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*. 511-514.

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.

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.

Bradley, A. P. (1997). The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition* **30**. 1145-1159.

Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery* **2**(2). 121-167.

Cadiz, J., Narin, A., Jancke, G., Gupta, A. and Boyle, M. (2004). Exploring PC-Telephone Convergence with the Enhanced Telephony Prototype. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*. 215-222.

Cutrell, E., Czerwinski, M. and Horvitz, E. (2001). Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. *Proceedings of Human-Computer Interaction (INTERACT 2001)*. 263-269.

Czerwinski, M., Cutrell, E. and Horvitz, E. (2000a). Instant Messaging and Interruptions: Influence of Task Type on Performance. *Proceedings of the Australian Conference on Computer-Human Interaction (OZCHI 2000)*. 356-361.

Czerwinski, M., Cutrell, E. and Horvitz, E. (2000b). Instant Messaging: Effects of Relevance and Time. *Proceedings of the British HCI Group Annual Conference (HCI 2000)*. 71-76.

Czerwinski, M., Horvitz, E. and Wilhite, S. (2004). A Diary Study of Task Switching and Interruptions. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*. 175-182.

Dabbish, L. and Kraut, R. E. (2004). Controlling Interruptions: Awareness Displays and Social Motivation for Coordination. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*. 182-191.

Dahlbäck, N., Jönsson, A. and Ahrenberg, L. (1993). Wizard of Oz Studies - Why and How. *Proceedings of the International Conference on Intelligent User Interfaces (IUI 1993)*. 193-200.

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.

Dey, A. K., Salber, D. and Abowd, G. D. (2001). A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal* **16**(2-4). 97-166.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*: John Wiley and Sons.

Dunning, D., Griffin, D. W., Milojkovic, J. D. and Ross, L. (1990). The Overconfidence Effect in Social Prediction. *Journal of Personality and Social Psychology* **58**(4). 568-581.

Edwards, W. K., Bellotti, V., Dey, A. K. and Newman, M. W. (2003). Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2003)*. 297-304.

Fayyad, U. M. and Irani, K. B. (1993). Multi-Interval Discretization of Continuous Valued Attributes for Classification Learning. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1022-1027.

Feldman-Barrett, L. and Barrett, D. J. (2001). An Introduction to Computerized Experience Sampling in Psychology. *Social Science Computer Review* **19**(2). 175-185.

Fogarty, J., Au, C. and Hudson, S. E. (2006). Sensing from the Basement: A Feasibility Study of Unobtrusive and Low-Cost Home Activity Recognition. *Submitted for Review*.

Fogarty, J., Baker, R. S. and Hudson, S. E. (2005a). Case Studies in the use of ROC Curve Analysis for Sensor-Based Estimates in Human Computer Interaction. *Proceedings of Graphics Interface (GI 2005)*. 129-136.

Fogarty, J., Hudson, S. and Lai, J. (2004a). 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.

Fogarty, J., Hudson, S. E., Atkeson, C. G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J. C. and Yang, J. (2005b). Predicting Human Interruptibility with Sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)* **12**(1). 119-146.

Fogarty, J., Ko, A. J., Aung, H. H., Golden, E., Tang, K. P. and Hudson, S. E. (2005c). Examining Task Engagement in Sensor-Based Statistical Models of Human Interruptibility. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*. 331-340.

Fogarty, J., Lai, J. and Christensen, J. (2004b). Presence versus Availability: The Design and Evaluation of a Context-Aware Communication Client. *International Journal of Human-Computer Studies (IJHCS)* **61**(3). 299-317.

Fraser, N. M. and Gilbert, G. N. (1991). Simulating Speech Systems. *Computer Speech and Language* **5**(1). 81-99.

Freund, Y. and Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences* **55**(1). 119-139.

Gajos, K. and Weld, D. S. (2005). Preference Elicitation for Interface Optimization. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2005)*. 173-182.

Gibbs, W. (2005). Considerate Computing. *Scientific American*. **292**. 55-61.

Gillie, T. and Broadbent, D. (1989). What Makes Interruptions Disruptive? A Study of Length, Similarity, and Complexity. *Psychological Research* **50**. 243-250.

Goffmann, E. (1982). On Facework. *Interaction Ritual*. E. Goffmann. New York, Random House**:** 5-45.

Green, D. and Swets, J. (1966). *Signal Detection Theory and Psychophysics*. New York, John Wiley and Sons. 45-49.

Hand, D. J. and Till, R. J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning* **45**(2). 171-186.

Hanley, J. A. and McNeil, B. J. (1982). The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve. *Radiology* **143**. 29-36.

Hatch, M. J. (1987). Physical Barriers, Task Characteristics, and Interaction Activity in Research and Development Firms. *Administrative Science Quarterly* **32**. 387-399.

Hess, S. M. and Detweiler, M. (1994). Training to Reduce the Disruptive Effects of Interruptions. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. 1173-1177.

Ho, J. and Intille, S. S. (2005). Using Context-Aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*. 909-918.

Horvitz, E. (1999). Principles of Mixed-Initiative User Interfaces. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1999)*. 159-166.

Horvitz, E. and Apacible, J. (2003). Learning and Reasoning about Interruption. *Proceedings of the International Conference on Multimodal Interfaces (ICMI 2003)*. 20-27.

Horvitz, E., Apacible, J., Subramani, M., Sarin, R., Koch, P., Cadiz, J., Narin, A. and Rui, Y. (2003a). Experiences with the Design, Fielding, and Evaluation of a Real-Time Communications Agent. *Microsoft Research Technical Report MSR-TR-2003-98*. ftp://ftp.research.microsoft.com/pub/tr/TR-2003-98.pdf.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D. and Rommelse, K. (1998). The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *Proceedings of the Conference on Uncertainty and Artificial Intelligence (UAI 1998)*. 256-265.

Horvitz, E., Jacobs, A. and Hovel, D. (1999). Attention-Sensitive Alerting. *Proceeding of the Conference on Uncertainty and Artificial Intelligence (UAI 1999)*. 305-313.

Horvitz, E., Kadie, C., Paek, T. and Hovel, D. (2003b). Models of Attention in Computing and Communication: From Principles to Applications. *Communications of the ACM* **46**(3). 52-59.

Horvitz, E., Koch, P. and Apacible, J. (2004). BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*. 507-510.

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.

Horvitz, E., Koch, P., Sarin, R., Apacible, J. and Subramani, M. (2005). Bayesphone: Precomputation of Context-Sensitive Policies for Inquiry and Action in Mobile Devices. *Proceedings of User Modeling (UM 2005)*. 251-260.

Huang, X., Alleva, F., Hon, H.-W., Hwang, M.-Y. and Rosenfeld, R. (1993). The Sphinx-II Speech Recognition Systems: An Overview. *Computer Speech and Language* **7**(2). 137-148.

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.

Hudson, S. E., Fogarty, J., Atkeson, C. G., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J. C. 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.

Intille, S. S., Larson, K., Beaudin, J. S., Nawyn, E., Munguia Tapia, E. and Kaushik, P. (2005). A Living Laboratory for the Design and Evaluation of Ubiquitous Computing Technologies. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*. 1941-1944.

Iqbal, S. T., Adamczyk, P. D., Zheng, X. S. and Bailey, B. P. (2005). Towards an Index of Opportunity: Understanding Changes in Mental Workload During Task Execution. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*. 311-230.

Jones, E. E. and Nisbett, R. E. (1971). *The Actor and the Observer: Divergent Perceptions of the Causes of Behavior*. New York: General Learning Press.

Kendon, A. and Ferber, A. (1973). A Description of Some Human Greetings. *Comparative Behavior and Ecology of Primates*. R. P. Michael and J. H. Crook. London, Academic Press**:** 591-668.

Kern, N., Antifakos, S., Schiele, B. and Schwaninger, A. (2004). A Model for Human Interruptability: Experimental Evaluation and Automatic Estimation from Wearable Sensors. *Proceedings of the IEEE International Symposium on Wearable Computing (ISWC 2004)*. 158-165.

Kern, N. and Schiele, B. (2003). Context-Aware Notification for Wearable Computing. *Proceedings of the IEEE International Symposium on Wearable Computing (ISWC 2003)*. 223-230.

Klemmer, S. R., Sinha, A. K., Chen, J., Landay, J. A., Aboobaker, N. and Wang, A. (2000). SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2000)*. 1-10.

Ko, A. J. and Myers, B. (2004). A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems. *Journal of Visual Languages and Computing* **16**(1-2). 41-84.

Kohavi, R. and John, G. H. (1997). Wrappers for Feature Subset Selection. *Artificial Intelligence* **97**(1-2). 273-324.

LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G. and Schilit, B. N. (2005). Place Lab: Device Positioning Using Radio Beacons in the Wild. *Proceedings of the International Conference on Pervasive Computing (Pervasive 2005)*. 116-133.

Langley, P. and Sage, S. (1994). Induction of Selected Bayesian Classifiers. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI 1994)*. 399-406.

Larson, R. and Csikszentmihalyi, M. (1983). The Experience Sampling Method. *New Directions for Methodology of Social and Behavioral Science* **15**. 41-56.

Lefcourt, H. M. and Martin, R. A. (1986). *Humor and Life Stress: Antidote to Adversity*. New York: Springer/Verlag.

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.

Liao, L., Fox, D. and Kautz, H. (2005). Location-Based Activity Recognition Using Relational Markov Networks. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 773-778.

Liu, H., Hussain, F., Tan, C. L. and Dash, M. (2002). Discretization: An Enabling Technique. *Journal of Data Mining and Knowledge Discovery* **6**(4). 393-423.

Lu, L., Zhang, H. and Jiang, H. (2002). Content Analysis for Audio Classification and Segmentation. *IEEE Transactions on Speech and Audio Processing* **10**(7). 504-516.

Mark, G., Gonzalez, V. M. and Harris, J. (2005). No Task Left Behind? Examining the Nature of Fragmented Work. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005)*. 321-330.

Markovitch, S. and Rosenstein, D. (2002). Feature Generation Using General Constructor Functions. *Machine Learning* **49**(1). 59-98.

Maulsby, D., Greenberg, S. and Mander, R. (1993). Prototyping an Intelligent Agent Through Wizard of Oz. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1993)*.

McCrickard, D. S., Chewar, C. M., Somervell, J. P. and Ndiwalana, A. (2003). A Model for Notification Systems Evaluation - Assessing User Goals for Multitasking Activity. *ACM Transactions on Computer-Human Interaction (TOCHI)* **10**(4). 312-338.

McFarlane, D. C. (1999). Coordinating the Interruption of People in Human-Computer Interaction. *Proceedings of Human-Computer Interaction (INTERACT 1999)*. 295-303.

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.

McFarlane, D. C. and Latorella, K. A. (2002). The Scope and Importance of Human Interruption in Human-Computer Interaction Design. *Human-Computer Interaction* **17**(1). 1-61.

Metz, C. E. (1978). Basic Principles of ROC Analysis. *Seminars in Nuclear Medicine* **8**(4). 283-298.

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.

Mitchell, T. M. (1997). *Machine Learning*: McGraw-Hill.

Miyata, Y. and Norman, D. A. (1986). Psychological Issues in Support of Multiple Activities. *User Centered System Design*. D. A. Norman and S. W. Draper. Hillsdale, NJ, Lawrence Erlbaum**:** 265-284.

Morimoto, C., Koons, D., Amir, A. and Flickner, M. (2000). Pupil Detection and Tracking Using Multiple Light Sources. *Image and Vision Computing* **18**(4). 331-335.

Munguia Tapia, E., Intille, S. S. and Larson, K. (2004). Activity Recognition in the Home Using Simple and Ubiquitous Sensors. *Proceedings of the International Conference on Pervasive Computing (Pervasive 2004)*. 158-175.

Mynatt, E. and Tullio, J. (2001). Inferring Calendar Event Attendance. *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2001)*. 121-128.

Nagel, K. S., Hudson, J. M. and Abowd, G. (2004). Predictors of Availability in Home Life Context-Mediated Communication. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2004)*. 497-506.

O'Conaill, B. and Frohlich, D. (1995). Timespace in the Workplace: Dealing with Interruptions. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1995)*. 262-263.

Oliver, N. and Horvitz, E. (2003). Selective Perception Policies for Guiding Sensing and Computation in Multimodal Systems: A Comparative Analysis. *Proceedings of the International Conference on Multimodal Interaction (ICMI 2003)*. 36-43.

Oliver, N., Horvitz, E. and Garg, A. (2002). Layered Representations for Recognizing Office Activity. *Proceedings of the International Conference on Multimodal Interaction (ICMI 2002)*. 3-8.

Osofsky, J. D., Ed. (1979). *Handbook of Infant Development*. New York, John Wiley & Sons, Inc.

Patterson, D. J., Liao, L., Fox, D. and Kautz, H. (2003). Inferring High-Level Behavior from Low-Level Sensors. *Proceedings of the International Conference on Ubiquitous Computing (UbiComp 2003)*. 73-89.

Perlow, L. A. (1999). The Time Famine: Toward a Sociology of Work Time. *Administrative Science Quarterly* **44**(1). 57-81.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*: Morgan Kaufmann.

Robertson, T. J., Prabhakararao, S., Burnett, M., Cook, C., Ruthruff, J. R., Beckwith, L. and Phalgune, A. (2004). Impact of Interruption Style on End-User Debugging. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2004)*. 287-294.

Sackett, G. P., Ed. (1978). *Observing Behavior, Vol. II: Data Collection and Analysis Methods*. Baltimore, University Park Press.

Schilit, B. N., LaMarca, A., Borriello, G., Griswold, W. G., McDonald, D., Lazowska, E., Balachandran, A., Hong, J. I. and Iverson, V. (2003). Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative. *Proceedings of the ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003)*. 29-35.

Schmidt, A. (2000). Implicit Human Computer Interaction Through Context. *Personal Technologies* **4**(2). 191-199.

Schmidt, A., Takaluoma, A. and Mäntyjärvi, J. (2000). Context-Aware Telephony Over WAP. *Personal and Ubiquitous Computing* **4**(4). 225-229.

Seshadri, S. and Shapira, Z. (2001). Managerial Allocation of Time and Effort: The Effects of Interruptions. *Management Science* **47**(5). 647-662.

Shell, J. S., Selker, T. and Vertegaal, R. (2003). Interacting with Groups of Computers. *Communications of the ACM*. **46**. 40-46.

Shrauger, J. S. and Osberg, T. M. (1981). The Relative Accuracy of Self-Predictions and Judgments by Others in Psychological Assessment. *Psychological Bulletin* **90**(2). 322-351.

Tang, J. C., Yankelovich, N., Begole, J., Van Kleek, M., Li, F. and Bhalodia, J. (2001). ConNexus to Awarenex: Extending Awareness to Mobile Users. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2001)*. 221-228.

Tullio, J., Goecks, J., Mynatt, E. D. and Nguyen, D. H. (2002). Augmenting Shared Personal Calendars. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2002)*. 11-20.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American* **265**(3). 66-75.

Whittaker, S. J., Frohlich, D. and Daly-Jones, O. (1994). Informal Workplace Communication: What is it Like and How Might We Support it? *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 1994)*. 131-137.

Wilson, D. H. and Atkeson, C. G. (2005). Simultaneous Tracking and Activity Recognition (STAR) Using Many Anonymous, Binary Sensors. *Proceedings of the International Conference on Pervasive Computing (Pervasive 2005)*. 62-79.

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*: Morgan Kaufmann.

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.

Zeigarnik, B. (1927/1938). On Finished and Unfinished Tasks. *A Source Book of Gestalt Psychology*. W. D. Ellis. New York, Harcourt Brace**:** 300-314.