Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

TITLE: Providing proactive support for task and interrupt management

SUBMITTED BY: Max Van Kleek
15 Pearl St. Apt #15
Cambridge, MA 02139

(SIGNATURE OF AUTHOR)

DATE OF SUBMISSION: January 23, 2006
EXPECTED DATE OF COMPLETION: June 2007
LABORATORY: Computer Science and Artificial Intelligence Laboratory

BRIEF STATEMENT OF THE PROBLEM:

This proposal describes an investigation into ways by which software can proactively support a user's task and time management practices within their desktop environments. The proposed approach surrounds methods that can infer associations between actions taken by the user within their applications, and descriptions of tasks on their pending to-do list. Once inferred, we explore how an augmented task management tool can employ such associations to help people more effectively budget their time and attentional resources, and in particular help them handle frequent interruptions. We also address user interface design implications and propose interface designs for proactive task management applications. Evaluation of each aspect will be done by prototyping the concepts and algorithms evaluated, gathering quantitative performance metrics from data gathered by users, and through qualitative user evaluations of the user interface.

Massachusetts Institute of Technology
Department of Electrical Engineering
and Computer Science
Cambridge, Massachusetts 02139

Doctoral Thesis Supervision Agreement

TO: Department Graduate Committee
FROM: Dr. Howard E. Shrobe

The program outlined in the proposal:

TITLE: Providing proactive support for task and interrupt management
AUTHOR: Max Van Kleek
DATE: January 23, 2006

is adequate for a Doctoral thesis. I believe that appropriate readers for this thesis would be:

READER 1: Professor Robert C. Miller
READER 2: Professor Michael Collins

Facilities and support for the research outlined in the proposal are available. I am willing to supervise the thesis and evaluate the thesis report.

SIGNED: _____
PRINCIPAL INVESTIGATOR
MIT CSAIL

DATE: _____

Comments:

_____

_____

_____

Massachusetts Institute of Technology
Department of Electrical Engineering
and Computer Science
Cambridge, Massachusetts 02139

Doctoral Thesis Reader Agreement

TO: Department Graduate Committee
FROM: Professor Robert C. Miller

The program outlined in the proposal:

TITLE: Providing proactive support for task and interrupt management
AUTHOR: Max Van Kleek
DATE: January 23, 2006
SUPERVISOR: Dr. Howard E. Shrobe
OTHER READER: Professor Michael Collins

is adequate for a Doctoral thesis. I am willing to aid in guiding the research and in evaluating the thesis report as a reader.

SIGNED: _____
ASSISTANT PROFESSOR OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

DATE: _____

Comments:

Massachusetts Institute of Technology
Department of Electrical Engineering
and Computer Science
Cambridge, Massachusetts 02139

Doctoral Thesis Reader Agreement

TO: Department Graduate Committee
FROM: Professor Michael Collins

The program outlined in the proposal:

TITLE: Providing proactive support for task and interrupt management
AUTHOR: Max Van Kleek
DATE: January 23, 2006
SUPERVISOR: Dr. Howard E. Shrobe
OTHER READER: Professor Robert C. Miller

is adequate for a Doctoral thesis. I am willing to aid in guiding the research and in evaluating the thesis report as a reader.

SIGNED: _____
ASSOCIATE PROFESSOR OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

DATE: _____

Comments:

1 Introduction

Technology has placed new demands on time management and task organization techniques. Recent advances in computer and networking technology have made possible instant communication and access to vast wealths of information. While this new immediacy of access and communication, along with powerful new productivity software, has, in theory, facilitated increased productivity, it has, in fact, resulted in an accompanying flood of new projects, roles, responsibilities and social obligations, pertaining to both work and to life outside of work. At the same time, the number of administrative support staff members formerly dedicated to helping individuals manage their daily schedules and responsibilities at work has dramatically decreased, having been reassigned to managing large groups of people rather than individuals. This has left the individuals to determine for themselves how to manage all of their new responsibilities.

To keep track of these numerous tasks and responsibilities, most knowledge workers have traditionally used task management schemes that rely upon a combination of physical and digital tools, such as paper organizers, post-it notes, a proliferation of to-do lists, as well as online calendars, e-mail inboxes [2], and the like [3]. Obviously, the effectiveness of an individual's task-management strategy was highly variable, largely dependent on how much effort was put into manually maintaining and updating each system. Compounding the problem, most people have shown a preference for expending an absolute minimum amount of time and effort in managing their task-lists[1]. As a consequence, they have been forced to internalize their task management schemes, relying less on external tools. In turn, internalizing task-lists has led to increased stress in the workplace, as individuals forgot or neglected important tasks.

Another limitation of existing task management strategies is that they have not incorporated a means of coping with the frequent daily interruptions of the modern work environment. People's work patterns today can be highly fragmented, and this has been demonstrated to be both inefficient and the source of significant stress in the workplace. Frequent interruptions caused people to lose track of what they were working on, to repeat work that they may have already done, and even to forget tasks entirely. Failures in personal task management schemes such as these have resulted in workers feeling overwhelmed by their work, with their inability to fulfill all of their responsibilities.

This thesis investigates the question of how to build automatic and semi-automatic systems that can take a proactive part in supporting the daily time and task management activities for people, an efficient, automatic version of the personal administrative assistant of the past. These systems will prove more effective than current generation of digital time-management tools, since they will not require a constant, disciplined commitment to be manually updated. To accomplish this goal, the main technical challenge will to building algorithms that can automatically observe the user's actions as they use their computers, learn associations between their tasks and the actions taken, and finally, infer a

variety of additional information about the actions being taken, the tasks being worked on, and the user's overall work patterns.

Automatically inferring the user's tasks through observation of their actions is difficult for a number of reasons. First, the user's actions taken in applications software tend to be ambiguous, since modern applications software are largely direct-manipulation oriented [27] and exhibit little clear task structure. Thus, by observing only the user's activities, it is not always clear what the user is trying to accomplish. Second, due to the limits of instrumentation it may not be possible to fully observe what the user is doing, but instead only to gain an abstract, first-order, "noisy" approximation of the actions being carried out. Third, rather than being able to have rich, formal descriptions of the tasks needing to be performed, the system will likely have to work with extremely impoverished, unstructured descriptions, in order to reasonably expect people to be willing to generate these descriptions for their tasks. Finally, there is probably a large degree of variability across how individuals execute their particular types of tasks.

To address these issues there are several potential promising hypotheses and approaches. The first hypothesis addresses *task regularity*, the recognition that there is similarity among the tasks performed by the user over time, which may then allow systems to recognize new tasks and predict how they will be accomplished. The second hypothesis focus on *action regularity*, the tendencies of a user to solve problems and accomplish tasks in a similar fashion each time they occur. If an algorithm could discern both of these types of patterns, it would be able to predict which tasks were likely to occur, and moreover, how the user was likely to respond. Therefore, we will devise methods that take the approach of individual, personalized modeling of users' tasks and actions, in order, conversely, to identify actions corresponding to tasks.

To evaluate the suitability and value of the information inferred by these methods, the second half of this thesis will work towards building a prototype application, an *automatic task list and interrupt manager*, or *ATLIM*. This application will function like a familiar desktop to-do list and calendaring application, but will also incorporate a number of unique features. These include being able to identify what task the user is working on at any particular moment, keeping track of the user's tasks as they change focus, and managing incoming interruptions such as instant messenger requests and e-mails. By incorporating these new methods in an actual application, we will be able to obtain feedback directly from users regarding what they consider the system's strengths and weaknesses. By incorporating these users' responses, we will aim to achieve a system that helps people better manage their tasks and time.

2 Background

Recent ethnographic studies have examined the task management practices of knowledge workers in their daily routines at work. Observations derived from these studies revealed that effective task management is crucial for most knowledge workers, in order for them to manage the sheer number and complexity

of the tasks required of them. These observations, as outlined in this section, identify a number of problems that form the basis of the design of an ATLIM, as proposed in this paper.

2.1 Task management practices in the workplace

Bellotti, et al., conducted a study [1] to identify what aspects of task management needed improvement. Her team made a list of observations during the study, summarized briefly as follows:

1. *People expend minimal time and effort adding items to their to-do lists; lists are often incomplete (mean: 10%), and descriptions terse.*
2. *People used a large assortment (mean: 11) of physical and digital artifacts to remind them of their to-dos. These often include "implicit reminders" requiring little or no effort to set up.*
3. *People like pipelining tasks when convenient, but are poor at it.*
4. *People like having ready access to the list of their commitments.*
5. *People are good at quickly assessing task priority.*
6. *Many tasks do not get done.*

Based on her team's observations, Bellotti prescribed a set of requirements for the design of a task-management tool. This set of design criteria, which we have adopted as the basis for our design of the ATLIM, includes recommendations for making task list input as easy and fast as possible, the use of *task histories* to allow the user to easily recall the state of tasks, and rendering task lists in a way that the entire list can constantly remain along the periphery of the user's view of their workspace.

2.2 The effect of interruptions

In a separate study [21], Mark, et al., has carefully analyzed how workers spend their time in a typical workday. Mark's team has discovered that due to a number of reasons, most people's typical work patterns are highly fragmented, resulting from a nearly constant stream of interruptions. A selection of her findings are summarized as follows:

1. *A majority of a person's tasks (mean: 54%) are interrupted each day.*
2. *Workers rarely spend more than a few minutes on any one task ("working sphere") before switching to another.*
3. *Users dislike disruptions that cause them to switch working spheres.*
4. *The majority of interrupts during the day were for personal, "metawork", or other purpose, and did not pertain to workers' central or peripheral tasks.*

5. *People have trouble resuming interrupted tasks.*

6. *Interrupted tasks are often not resumed until much later in the day, and sometimes even later.*

7. *Even without explicit interruption, people switch tasks regularly.*

Instead of counting tasks as individual actions, she created the notion of *working sphere* to refer to the set of actions that contributed to the same high-level objective. Mark then classified interrupts experienced by their subjects as belonging to either the from same working sphere as the user's active task, or from a separate working sphere. Mark's team also distinguished *external* interrupts, where some event occurred that caused a user to switch working spheres, from *internal* interrupts, which occurred whenever a person decided to switch tasks and no such event occurred.

Using this classification scheme, Mark made several important observations. The first was that interrupts pertaining to a different working sphere than the user's active task caused the greatest disruption in work flow, while interrupts that coincided with the user's central working sphere were not only welcomed, but often helped the user accomplish their task. Therefore, filtering these latter types of interruptions would be detrimental. However, a vast majority of interruptions that occurred throughout their subjects' workdays were not relevant to their central working sphere, and most were due to personal, "metawork" or other, non-primary task related working spheres. Therefore, a majority of interruptions were of the type that were highly disruptive to the central task.

Second, while the percent of internal interrupts taken by users varied, most people frequently switched tasks even in the absence of external interrupts. The team concluded that this was because people were constantly monitoring and re-prioritizing their tasks based upon the dynamics of their workplace, switching whenever another task grew higher in importance. This observation was supported in Bellotti's study, which observed that people liked to constantly be aware of their task lists so they could choose their activities from each moment to the next. This constant switching of tasks tended to incur for the user, a penalty in time and effort, due to the need for them to re-orient themselves to their interrupted work state whenever they resumed a task.

The final important observation came from comparing tasks which were not interrupted with those that were more fragmented. Tasks that were less interrupted were generally returned to more rapidly, and completed more immediately, while those that were more fragmented were returned to much later.

These observations indicate that work fragmentation is indeed a problem, whether interrupts were advantageous or detrimental was dependent on a number of factors, mostly connected with how relevant the interrupt is to the user's active task. This forms the motivation for our task-based interrupt filtering strategy, which will be described in section 4.2.1.

2.3 Previous approaches

The deficiencies in people's own task-list management schemes and interrupt management policies revealed by Bellotti and Mark's studies indicate the need for better tools to help support these activities. Several research efforts besides the discussed work by Bellotti introduced designs for addressing task and interrupt management challenges in various HCI-oriented ways.

A majority of these approaches sought to help users by reducing the burden of switching tasks. Research has primarily followed two main approaches to solving this problem: facilitating window/desktop layout for running applications, and second, a re-consideration of file/data-space around notion of tasks or activities instead of folders and databases.

Investigations of the former approach include work by MacIntyre et al. designed *Kimura*, a system that allowed users to visually organize their desktop on a per-task basis, utilizing peripheral displays to maintain awareness of these desktops when extra screens were available [20]. Kimura aimed to help remind users of other pending tasks, and to expedite task switching by helping them instantly restore previous desktop states. Czerwinski's group at Microsoft [7] proposed extensions known as GroupBar [28] and Scalable Fabric [26], for organizing open windows around relevant tasks.

For the latter approach, Kaptelinin introduced *UMEA*, a system that tracked documents opened in Microsoft Office during the execution of a particular task [17]. These documents were later displayed alongside the task for easier retrieval. More recently, Geyer et al.'s designs for a shared-object workspace for Lotus Notes, introduced a re-organization of Notes databases around *activities* which were more easily shared [11].

These approaches introduced the notion of organizing resources around representations of users' tasks, instead of applications or filesystem resources. However, with the exception of the UMEA work (which used simple rules to infer associations), they were entirely manual, requiring the user to take explicit initiative and action to maintain. On one hand, this gave the user complete control over how to arrange and organize their resources; on the other hand, it required time, effort and dedication, essentially comprising yet another task which the user must regularly update. We imagine a system that preserves this notion of activity, but which could proactively support users' organization schemes by automatically taking actions in various ways. Instead of imposing some preprogrammed organizational scheme on the user, however, we propose that the system should *learn* the user's organization scheme through observation.

A number of other researchers have very recently started to examine methods that could help make this possible; in particular, *TaskTracer* by Stumpf et al [9], and *SWISH* by Oliver et al. [24], have worked on automatically inferring the task or activity that best explain a user's actions. Similarly, Drezde et al. [22] demonstrated how similar techniques can be used to categorize e-mails by activities. These experiments, consisting of applying statistical learning techniques to infer the user's activity from both content and interaction-based features, yielded very promising results. However, these results are still very

preliminary, and leave open two major questions: first, whether the performance of their methods is adequate to be usable for a larger class of users in a real-world application; and second, whether the methods they employed can be generalized to both new interactions *and* new task/activity descriptions. These two questions are qualitatively more difficult than the problems posed in these experiments, and are two of the open questions we would like to address in this thesis.

3 Proposed approach

This project seeks to design a new type of task-management system that can observe the user, and take action to *proactively support* their task and interrupt management schemes. Unlike previous approaches that require the user to entirely manually take the initiative to update their to-do lists, our approach focuses on ways that systems can start to automatically take care of task-management activities while the user stays focused on the tasks at hand.

In designing a new application which uses methods from artificial intelligence to take proactive action on behalf of the user, it is first necessary to identify exactly what new capabilities are most needed from these methods. In order to determine this, we first identified the most promising scenarios of how proactive assistance would be most useful within an ATLIM, based on the conclusions drawn from Bellotti and Mark's studies. Then, using this list, it was possible to identify the core inferential capabilities that were necessary to enable each to happen, and then to identify the the set of capabilities that were most needed.

Based on this type of analysis, the following two inferential capabilities were identified as being linked to the largest set of potential ways that a system could proactively support the user's time and interrupt-management related activities:

1. *Identifying the user's active task based on their actions.*
2. *Identifying how interruptions pertain to a user's tasks.*

The following subsections, describe the primary set of activities that these two capabilities would support in an ATLIM.

3.1 Supporting the user's active task

The first way that an ATLIM should provide proactive assistance is in supporting the user's active task. This generally first involves identifying the task corresponding to the user's actions, and subsequently acting on this information in various ways. Several examples include:

1. *Providing applications with task context* - If the ATLIM could inform the user's active applications about the task the user is trying to accomplish, the applications could optimize the layout of their UI elements, and streamline their workflow for the particular task.

2. *Organizing access to task-related materials* - Most tasks are associated with a certain set of documents or resources which the person needs for accomplishing the task. Since users switch tasks frequently, they consequently expend inordinate amounts of time re-locating the sets of resources they need for each task. An ATLIM could help the user by automatically learning which documents a user needs for a particular task. This could be accomplished by examining the user's document access patterns when they are performing a particular task. Then, when the task is later resumed, or a new task of a similar nature is identified, the system could instantly retrieve and place within "close reach" the resources the user is likely to need.

3. *Keeping track of the task stack* - Since Mark's study revealed that people often lost track of what tasks they were involved with when an interruption occurred, the ATLIM could remind users about the tasks they were performing when they were interrupted, thereby reducing the disruptive effect of the interruption.

3.2 Supporting task-list management

As Bellotti described, the main reason that many task management schemes are abandoned is that people do not wish to expend the effort required to maintain the scheme. Thus, proactive assistance could be provided here to make the act of maintaining one's tasks much easier.

1. *Simplifying the addition and removal of items as they are completed* - An automated task list/interruption manager could ease the process of managing the contents of this list, and also make their task lists as complete as possible. Specifically, an ATLIM would work to greatly simplify the manual addition of tasks. An ATLIM would also help by automatically detecting and adding new tasks to their to-do list, classifying them appropriately into their proper categories. Similarly, an ATLIM tracking tasks being executed could automatically retire tasks that are likely to have been completed.

3.3 Interruption management

1. *Buffering irrelevant interruptions* - Mark's study revealed both perceived negative effects and a number of measurable consequences of interruptions that caused users to switch working spheres. These suggest that an ATLIM could provide a significant benefit if it were able to help mediate such interruptions. The obvious approach is to make these interruptions, in one way or another, arrive at a time that is not likely to force the user to switch working spheres. One way to achieve this would be to monitor the user's current working task, and then automatically postpone such incoming interruptions.

2. *Informing remote collaborators of task context* - Another approach is to provide appropriate social cues to local and distant collaborators regarding the user's task context, so that others might be able to more easily assess whether a particular time was appropriate for an interruption.

3.4 Learning task contexts and long-term work patterns

1. *Learning task settings* - Monitoring the user's work patterns would also allow the ATLIM to build long-term patterns of what kinds of tasks were performed at particular times and places, and how the user performed them throughout each day. This would make it possible for the ATLIM to identify contexts for when/where tasks need to be performed, such as "Home", "Office", or "on the Week-end", which it might use later to organize and help prioritize tasks.

2. *Monitoring task performance* - Other ways that an ATLIM could apply a model of the user's task flow over time would include determining how the user's performance varied under different conditions, including environmental conditions, or based upon how fragmented their work performances were. This might help make the user more aware of how they might wish to improve their workflow, such as by turning off sources of interruptions (i.e., making themselves unavailable) for the completion of certain important tasks, or by regulating other aspects of their environment.

3.5 Supporting task prioritization and selection

1. *Organizing tasks dynamically by relevance* - From the observations in Bellotti's study, we may infer that people tend to be adept at quickly deciding which task to select under a given circumstance. Therefore, an ATLIM should work to support users in making their own determination of what tasks need to be performed, rather than trying to automatically decide for the user what should be done at a particular moment. One way that an ATLIM would help users make such a judgement is to visually organize tasks as well as pending/postponed interruptions into relevant collections, or by other metrics such as urgency. Moreover, as the Bellotti study ascertained, people to be constantly aware of their task list regardless of where they were. Thus, an ATLIM could provide a readily-accessed, consistent view across applications through all the digital mobile and other devices with which they interact.

2. *Predicting task duration* - Another way that an ATLIM would help people assess the relative priority of tasks is by projecting how long the task might take and displaying this to the user. This will be particularly helpful, since the amount of time to task completion will likely be a factor in choosing what tasks to perform. Further, since we infer from Bellotti's findings that

people are generally poor at estimating how long it may take them to do something, or recalling how long a task actually took, this information would be provided by the ATLIM.

4 Research challenges

Realizing the goals just described for an ATLIM presents numerous wide-reaching technical challenges, encompassing domain knowledge about a wide array of tasks, data mining interaction streams and sources of user information, and long-term modeling of users' workflow patterns. This section will attempt to further define the capabilities needed, and identify the associated technical challenges.

4.1 Identifying the user's active task

In this section, we describe the problem of designing an algorithm that can automatically (or semi-automatically) identify what tasks users are working on by observing their actions. In order to do so, we will first frame the problem by describing what we mean by "observe the user", then proceed to describing how we will represent a task, and, finally, we will describe various approaches to solving the actual task identification problem.

4.1.1 Action observations: Interaction traces filtered with domain knowledge

This section outlines the *sensor model* of the system, comprising user interaction traces acquired automatically by the ATLIM as the user goes about regular daily activities at their workstations.

One option would be to instrument the user's environment to capture raw, low-level events, such as key-presses and mouse movements, as well as low-level application events, such as button presses and menu actions. Previous efforts, such as by Horvitz et al. [13] for inferring interruptibility, have attempted inferring high-level user states using these low-level actions. However, since these low-level actions do not easily capture the semantics of the user's actions, (e.g.., keystrokes registered in a Word document window versus those registered typing in a command terminal correspond, semantically to different types of user actions), we will first try a slightly different approach, which initially pre-processes low-level input into a high-level action representation. In addition to differentiating dissimilar semantic categories of user action, this initial processing step works to consolidate semantically equivalent actions; for example, viewing the same web page using different web browsers should still be treated as the same action.

In our initial schema for user action, illustrated in Figure 2, we have abstracted user actions to one of five types: *C*ommunicating with someone, *R*eading

information, *E*diting/creating a document, *T*ouching (such as moving) a document, *S*witching to an application, and being *I*dle. The C, R, E, T, and S actions all accept parameter vectors, which modify the action; the *what:* parameter specifies the name of the object being viewed/edited/modified, etc.; the *about:* parameter is a feature vector "summarizing" the content being viewed/edited by the user. These abstract actions allow us to treat events such as opening a PDF, accessing a web page, or reading e-mail all as "Read" actions, while their corresponding *about:* parameter allows documents with different content to be distinguished. Thus, our simple, high-level action vocabulary provides access to basic user actions at a larger granularity than user interface events, in addition to application independence.

A technical challenge associated with analyzing interaction traces arises from acquiring these action traces from conventional desktop and mobile applications. Most of the current generation of applications provide extremely limited support for allowing external applications to access the user's actions. Moreover, even in cases where a means of such introspection is provided, these usually require programming for an application-specific API which cannot be re-used for other applications. Thus, our technique will be to build pluggable application-specific modules to interface with individual desktop applications where possible, and to have these modules translate the application-specific events into our action ontology. Actions detected by all modules will then be consolidated and output in a single stream.

4.1.2 Representations of Tasks

As will be described in the next section (4.1.3), the core of the algorithm will be dedicated to to identifying how actions contribute to certain tasks, even for new actions and tasks that have previously not been seen. In order for this to be feasible, the representation of both actions and tasks (introduced as $Desc(t_i)$) have to be sufficiently rich, so that the learning algorithm can generalize what it has learned to these properties of tasks and actions (instead of the identity of each of the actions themselves). The drawback of requiring rich representations of tasks, however, is that it adds to the overhead required of users to specify these tasks to the system; and as conveyed by Bellotti [1], people have little patience for adding tasks to their to-do lists. Thus, we will have to consider tradeoffs between richness of representation (which may provide greater task-identification accuracy), and overhead required of the user to update such a representation. In Bellotti's implications for task management design, she emphasized the need for task list managers to support "quick and easy input", requiring "no formal task description, categorization, or decomposition [... and to support descriptions at] any level of task abstraction for atomic task entries." However, providing user interface mechanisms and flexibility in the the process of specifying tasks may ease the burden somewhat on users to specify some structure in their task descriptions.

We will consider the following as an initial schema for representing tasks:

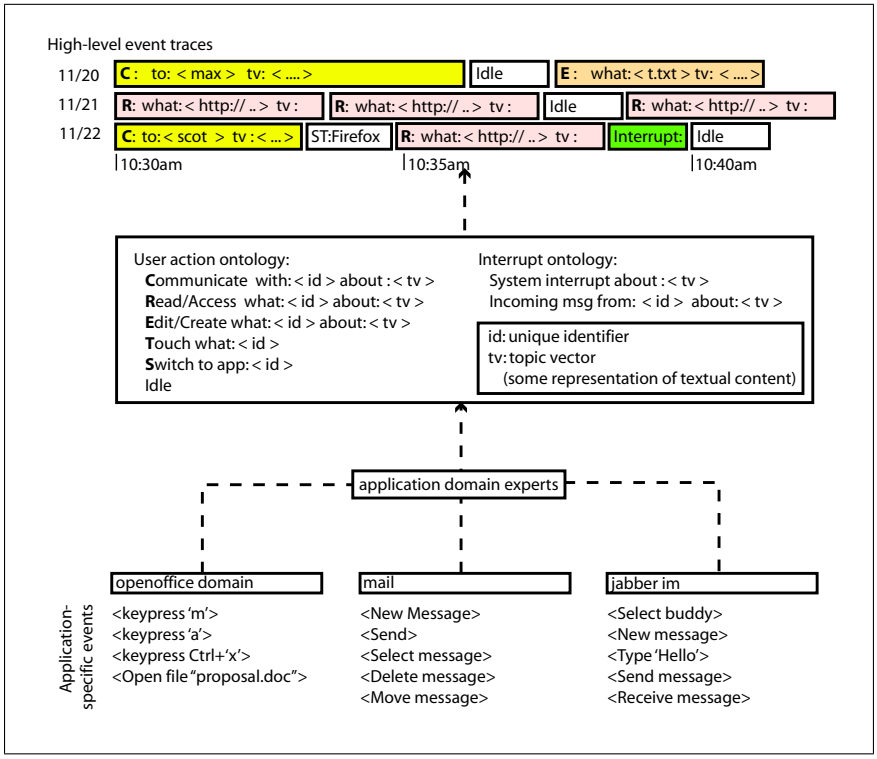

Figure 1. Events corresponding to user actions and interruptions first get abstracted into a high-level action/interruption ontology.

1. *Task description/tags* - Short freeform textual descriptions of the task
2. *Task type* - Allow users to specify semantic categories for tasks of their choice, such as "work", "home", "Client X"
3. *Priority and Deadline* - Rough value of "importance level" of this task to the user; Date/time when the task is due.
4. *Related people* - People who are relying on this task's completion, or who are required of this task
5. *Related documents* - Documents pertaining to the task

To provide the user with flexibility as recommended by Bellotti, most of these fields will be allowed arbitrary freeform textual descriptions; in addition, the user interface will provide quick access to formerly specified values. For specifying related documents, the ATLIM user interface will allow users to select either from a list of recently accessed documents in the filesystem, or to drag and drop files directly into the field. See section 4.3.2 for a more detailed description of our UI design.

As will be described (Section 5), our third phase of investigation will seek to improve this representation, based both on user feedback regarding whether they thought the process of describing tasks was decent or too onerous, as well as based on the task-action mapping algorithm's performance.

4.1.3 Mapping actions to tasks

Our challenge, thus, is to take our stream of user actions, described in terms of the user action schemas (4.1.1), and a set of known to-do items provided by the user in terms of task schemas (4.1.2), and to identify which actions contribute to which task. This problem can be treated in a number of different ways. These are considered as follows:

1. *Learning* - It may be possible for the system to learn correspondences between action traces and tasks through explicit training data from the user. This training data will consist of essentially, action traces with corresponding task descriptions from their to-do list. With enough training data, the system may be able to generalize characteristics of actions and descriptions which let the system identify when the user is performing a particular task. Depending on how similar we assume people to be, this learning could be done across users (if we assume similarity) or on an individual basis.

2. *Knowledge-based plan recognition* - If the task representation is viewed as representing a goal, and a library of (partial) plans constituting a mapping from sets of actions to goals or sub-goals is available, the problem of identifying what goal (task) the user is working on at any moment is analogous to that of *plan recognition* [4].

The former approach is more general and makes few prior assumptions about individuals, while the latter relies on specific knowledge about tasks and individuals' work practices. The previous approach is therefore more adaptive/accommodating to variations among users and work practices. Therefore, this learning-based approach is more attractive to us, at least initially, since we know little about our users' work practices. On the other hand, the primary risk is that these "pure" learning approaches is that they may require an infeasible amount of training data from the user in order to achieve adequate performance. If this turns out to be the case, we will try various approaches at incorporate knowledge to effectively "tune" models to individuals in order to improve performance. Potential approaches of doing this will be discussed in section 4.1.10.

4.1.4 Approach 1: A classical Bayesian learning formulation

In order for the system to learn associations between actions and tasks, let us first assume that the user is going to provide the system some assistance, by providing some training data. This training data will constitute labeled action

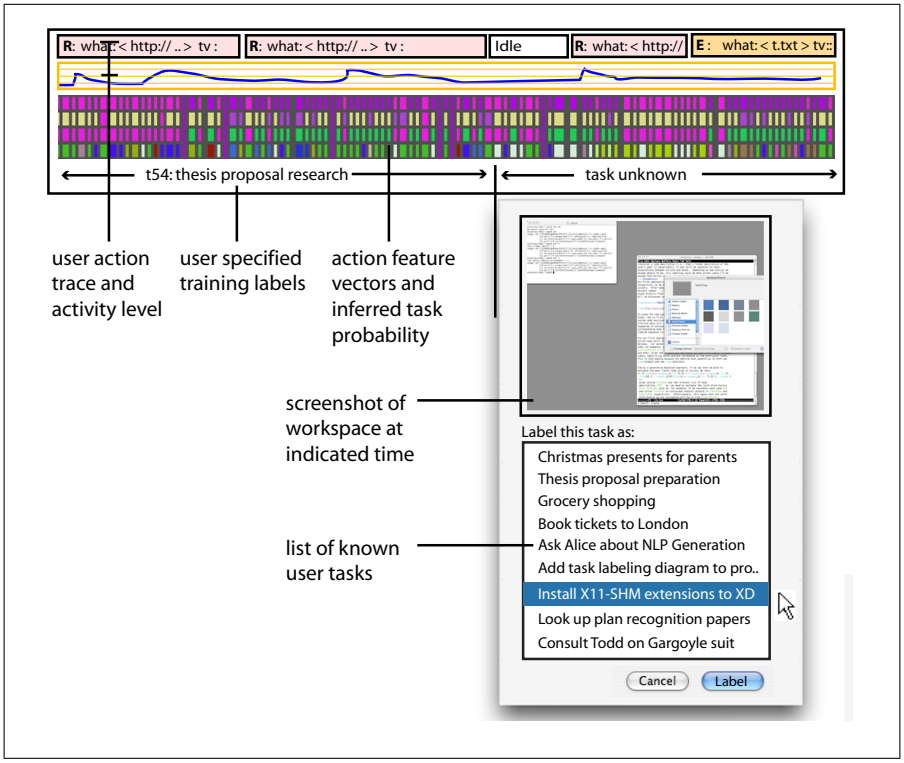

Figure 2. Mockup of ATLIM user interface for displaying the user's recent action history. The visualisation (top) displays the action classes ordered along a time-line, below which exists a a graph of the user's workstation activity level over time. Below this is a data grid using color coding to represent parameter values for the action classes, and the inferred most likely task at that moment. Hovering the mouse over the grid will reveal the meaning of each color. Users train the system by clicking on transition points between tasks, which causes the visible drop-down dialog box to appear. This dialog box contains a low-resolution screen shot captured at that transition point and a choice box containing the contents of their task-list.

streams; specifically, sequences of actions where each sequence is labeled with the corresponding task that the user was performing. An example such labeled sequence is illustrated in Figure 2.

For our first approach, let us consider training a machine using only action-task pairs for identifying to which task a particular action belongs. Our objective, thus, is to build a learning machine that can take, as examples, a set of action-task pairs, e.g.: $\{(\alpha_0, t_2), (\alpha_1, t_4), (\alpha_2, t_4), (\alpha_3, t_5)...\}$ and then, given new actions and new tasks not seen previously, produce labels identifying which actions correspond to the particular tasks. This is challenging because the machine must generalize to both new *tasks* and new *actions*.

Taking a generative Bayesian approach, if we say that we wish to estimate

the most likely task given an action, we have:

$$F(\alpha) = \arg\max_{t \in T} P(t|\alpha) = \arg\max_{t \in T} \frac{P(t,\alpha)}{P(\alpha)} = \arg\max_{t \in T} P(t,\alpha)$$

given action α and the relevant list of task descriptions, T. So, we need to estimate the joint distribution $P(t,\alpha)$, such as, for example, if we represent each task t and action α as continuous feature vectors in $\Re^m$ and $\Re^n$, respectively. Unfortunately, this means that the joint distribution we are estimating will be of size $\Re^{m+n}$, which will be extremely large and sparse, given that we wish to keep the training set size, $|T|$, as small as possible.

4.1.5 Approach 2: Using a discriminative classifier to reduce dimensionality

To get around the dimensionality explosion just described, we can employ methods designed for learning in *highly structured output spaces*, such as the modified perceptron method first considered for re-ranking parse-trees[5]. This method uses a feature kernel function, $\Phi(\alpha,t)$ which extracts features from both α and t, mapping each (α,t) pair to q-dimensional feature vector, $\phi \in \Re^q$. In general, $q < (m+n)$ because the q features can represent *joint features*, or features comparing features of α with those of t. An example of such a joint feature might be a value representing the dot-product between the term similarity vectors for the action and the description of the task:

$$\phi_i(\alpha_j, t_k) = \langle about(\alpha_j), Desc(t_k) \rangle$$

Thus, under this transformation, the feature space becomes not that of all possible tasks and actions, but that of the the relationship between tasks and actions.

In order to learn associations between actions and tasks under this framework, we can use the perceptron training algorithm modified by Collins [5]. This algorithm is illustrated in Figure 3. In this algorithm, we optimize the vector W of weights for the feature elements of $\Phi(\alpha,t)$ so that we most often choose the correct task. Collins provides proof of convergence for W under separable conditions in [6]. The number of iterations required for convergence depends on the separability of the data. We do not yet know how separable these action-task mapping problems are under typical conditions; this will largely depend on the choice of the feature kernel function, $\Phi(\alpha,t)$.

The choice of $\Phi(\alpha,t)$ is crucial for determining the performance of this approach. Thus, this will be the primary area of investigation for this experiment: *What choice of a joint feature function, $\Phi(\alpha,t)$, if any, will yield acceptable performance for finding action-task correspondence?* To investigate this question, we start with a basic set of features comparing shared fields in the action schema, specifically $what(\alpha)$ and $about(\alpha)$, with task fields in $Desc(t)$. This comparison may be done using textual comparison techniques such as Latent Semantic Analysis (LSA), [8], which has been applied to many problems involving "semantic" comparisons between textual passages. If such features are insufficient, we will next consider auxiliary context, such as those concerned with action sequences.

Inputs: Training set, (α_i, t_i) pairs for $i = 1..n$

1. Initialize $W \leftarrow 0$
2. Let $F(\alpha) = \arg\max_{t \in T} \langle \Phi(\alpha_i, t_i), W \rangle$
3. For $iteration = 1..MAXITERS$ and $i = 1..n$,
   Let $z_i = F(\alpha_i)$.
   if $(z_i \neq t_i)$ then $W \leftarrow W + \Phi(\alpha_i, t_i) - \Phi(\alpha_i, z_i)$
4. Output: W^*, the final value of W

The most likely task for a given action is then:
$F(\alpha) = \arg\max_{t \in T} \langle \Phi(\alpha_i, t_i), W^* \rangle$

Figure 3. Collins' modified perceptron learning algorithm in terms of tasks

4.1.6 Approach 3: Learning directly from action traces

In addition to the supervised-training based approaches just described, it may be possible to learn the user's work patterns and information about their ways of accomplishing tasks directly from action traces. One approach for doing this is to identify similarities among different actions, amounting to clustering actions into *action classes*. Clustering actions in this way will make it possible to use a finite "alphabet" of labels to represent the continuous, potentially infinite variations of actions generated by the user. These metrics could include similarity of the actions themselves, or similarity of their context, such as when, and how long, or where in action streams they typically occur.

Once such an action alphabet is identified, we could use any standard sequence-learning technique to identify structure from action sequences. One such technique would be to treat the sequence of actions as a stochastic process, and to make a Markov assumption to model action class transition probabilities. The resulting *markov chains* could be analyzed for states that exhibit periodicity, and positive recurrence, and then could be analyzed for their predictive power. Identifying such recurring states could be helpful for identifying which applications are used on a regular basis and task-independence. Furthermore, if chain is *reducible*, meaning that is not fully connected, the states in the chain could be partitioned into sets of *communicating classes*, or relatively self-contained clusters. These communicating classes could be indicative of the the user's task – which, if identified, could greatly simplify the process of learning mappings between tasks and action flows. A hypothetical example of how this might work is illustrated in Figure 4.

Based on Czerwinski's observations that people tend to exhibit *monitoring behaviors* (or rapidly switching between applications), it may be necessary to use Markov assumptions of order greater than 1 in order to capture users' task flows. We will examine how the order of the Markov process used affects the system's predictive performance[29].

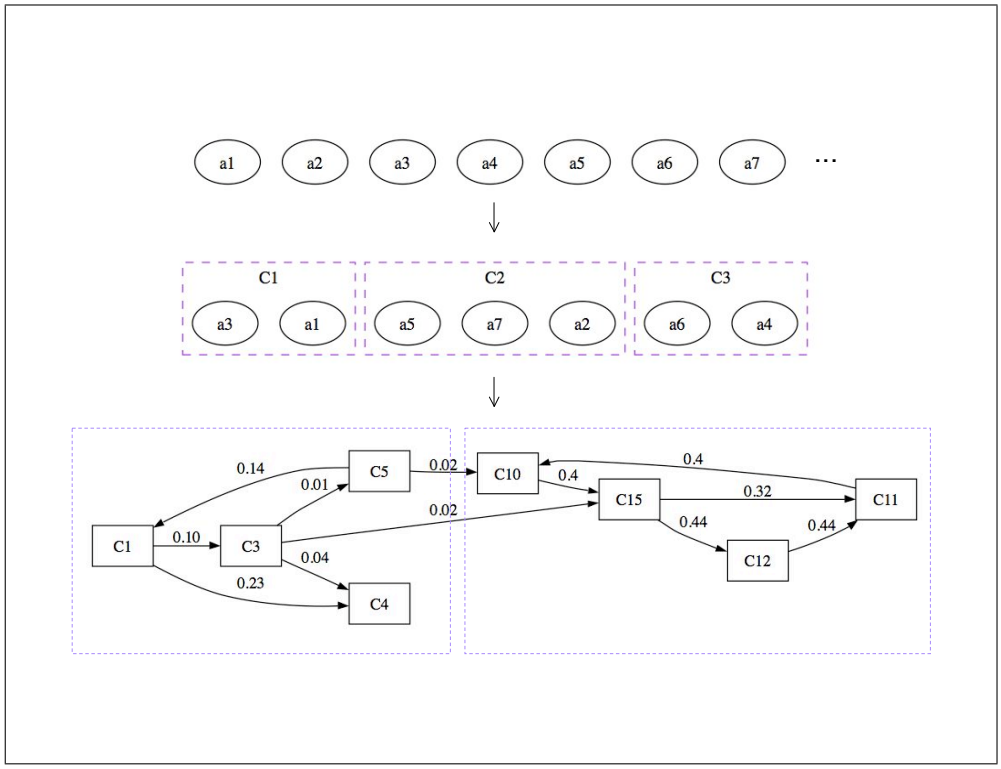

Figure 4. Example of analyzing action sequences; 1) actions are first clustered into action classes; 2) transition probabilities are learned among the action clusters. If the resulting markov chain can be partitioned into strongly-connected sets, this could be indicative of tasks.

4.1.7 Other approaches: Combining action transitions with action-task probabilities

As emphasized in the introduction of this section, the approaches outlined in sections 4.1.4, 4.1.5, and 4.1.6 not comprehensive, but are rather examples of the types of methods that we will try to apply to this problem. Until we have captured data and have a chance at evaluating these methods, it is difficult to discuss *a priori* the suitability to our particular task.

A final example of another type of approach which is promising by modeling both action sequences and action-task mappings are *Conditional Random Fields*, or *CRFs*, introduced in [19]. This technique, which learns the probability of an entire sequence of labels (in our case, tasks) conditioned on an entire sequence of observations (actions), without having to model the joint distribution, $P(\alpha, t)$ described in 4.1.4. This is done by modeling only the conditional distributions of the labels given the observations. Similar to $\Phi(\alpha, t)$ described in 4.1.5, CRFs rely on feature functions, but require two; one for transitions and one for observations. If the CRF is assumed to be a tree, The probability of a sequence of tags

given observations is an exponential of the inner product between these feature vectors and two parameter vectors to be learned. In our application, transition probabilities will correspond to the likelihood that one task follows another, while observation probabilities model the likelihood the user is working on a task, given the action.

4.1.8 Evaluating methods for learning correspondences between actions and tasks: How good is good enough ?

Each of the methods discussed above have advantages and disadvantages; direct estimation of the joint distribution is simple but maybe intractable due the high dimensionality of the space, the perceptron modeling technique is distribution-free but relies on an appropriate choice for a joint feature function, while unsupervised methods require little no training data, but rely on finding an appropriate choice for a clustering metric. Which of these methods (or others, such as Conditional Random Fields) work "best" for the application will depend on at least the following factors:

1. *Accuracy (Test Performance)* - How accurately the algorithm can predict the correct task given an action trace
2. *Amount of Training data / supervision required* - The amount of explicit user input, in the form of training, the system requires.
3. *Usability of task description* - Whether the description schema used by the method is agreeable to users. Since users will usually have to manually specify task descriptions often, it is important that the schema is not too onerous, and either flexible/accommodating to the user's style or of a fixed format that most users like.
4. *Efficiency* - Whether the algorithm can run on conventional desktop computers and not require an unreasonable amount of computational power or memory

The accuracy we require for the algorithm depends on how exactly the algorithm is to be used within the context of our proactive time-management application. Moreover, we realize that there is a large difference between demonstrating that an algorithm works *i.e.*, that is, that it achieves statistically performance over chance, from being accurate enough to be actually *useful* to our application. Since this is an entirely new recognition problem, it is difficult to predict how well the methods we have proposed will fare. Moreover, we do not yet know how effective refining the models and techniques will be towards yielding better results. Therefore, at this stage it is too preliminary for us to state a performance goal. We can only say that based on other, similar efforts such as in deducing users' interruptibility discussed later (section 4.2.1), initial attempts have yielded meager results. However, initial attempts such as ours

usually help the field to better understand the recognition problem at hand, and then to iteratively devise more appropriate techniques. Such patterns were observed historically with other, previous challenging recognition problems, such as speech recognition.

4.1.9 Assumptions made with learning techniques: Consistency and variability

The techniques just described in 4.1.3 make few prior assumptions regarding the user's work patterns, the resources they access, and how they describe the items on their to-do lists. Our approach, of choosing a large set of general features for these learning frameworks, leaves the task of uncovering the structure (if such a structure even exists) for mapping task descriptions to action traces to the learning algorithm. On one hand, this is advantageous because we do not yet know *a priori* whether such a common patterns exist, either on an individual basis or, even greater, across individuals. Therefore, our experimental framework aims to use machinery that starts uninformed about peoples' work practices, and learns from examples.

A fundamental assumption in this approach, however, is that people tend to go about their tasks in a way that exhibits some regularity. The existence of this regularity is not itself sufficient; this regularity also has to be *observable* by the learning algorithm through the from the representations that we choose for each for the action traces and task descriptions. The degree to which the regularity can be uncovered will largely determine the algorithm's performance; that is, how well the algorithm can predict when the user is working on particular task-item. Therefore, finding an appropriate representation for these items (including which features to examine from each) is critical for making the algorithm perform well.

We would also be interested in seeing whether patterns for mapping task descriptions to their actions are generalizable *across individuals*; this would enable models to be re-used and trained between people. Our intuition is that for any given individual, people tend to go about their tasks in a way that exhibits some regularity, while across individuals, there is likely a much larger degree of variability regarding how people work. This variability is influenced by such factors as personal style/preferences, and degree of familiarity with individual applications. However, if any common characteristics were found, parts of models could be shared across users, and training data provided by individuals could be shared to improve everyone's models. This could greatly benefit the training data problem, described next.

Since training data has to be manually provided by the user, it is likely that the amount of training data required to achieve levels of performance will likely be the deciding factor of whether the particular approach is feasible. This will lead us to select the methods that can achieve acceptable performance with the fewest number of examples. If there is only a small amount of training data available, compared to the dimensionality of the space, the *approximation error* for the system will be likely too great to be able to perform useful gen-

eralizations. One way to approach this is to *regularize* the learning algorithms, essentially to favor simpler solutions, given a limited training set. Strategies like this are described briefly next.

4.1.10 Strategies for improving generalization performance: Adding domain knowledge, classifier hierarchies, simplifying models

If our initial approaches with the above methods yield inadequate performance, we can try pursuing two complementary but different basic strategies: enriching the input (feature) representations, and, specializing the models/learning mechanisms. The former is a way of adding more information to the algorithm, while the latter essentially makes assumptions to reduce the "power" of the algorithm to make learning more tractable and generalization possible.

There are a couple possible ways to enrich the input representations. One would be to add explicit domain knowledge; for example, we could build a knowledge base categorizing a list of known resources and what sort of "activity" they represent; for example, "{dilbert, LEISURE}, { java.sun.com/apis, CODING }, ... ". Another approach is to perform such labeling by training lower-level classifiers. If these low-level classifiers could identify characteristics that do not vary significantly across users, they could be trained across users. Another example of such a low-level feature might be task transition detection. The output, then, of these classifiers could then be treated as features into the larger classifier, such as a log-linear model [18].

Similarly, there are different ways to specialize the learning task. One way is to make the assumption that each action trace was intended for exactly one of the user's known to-do items, simplifying the classification problem to choosing one among the $|T|$ items instead of having to make $|T|$ binary decisions.

Finally, as described in the last section, due to the limited amount of training data, we will likely have to restrict our models to limit the amount of over-fitting the algorithm performs on the training set. The process, known as *regularization*, can be accomplished in a variety of ways, depending on the learning mechanism employed.

4.2 Interruption management

4.2.1 Assessing interruption relevance by task relevance, instead of interruptibility

Several efforts have recently begun in designing systems that can automatically assess user *interruptibility*, or how open a user is to receive a disruption at a particular time. These investigations have so far largely involved seeking features which are most predictive of this abstract notion of interruptibility, and evaluating how these features perform when used in classifiers. An extensive early study by Forgarty et al. laid much of the groundwork in this subject, by considering a variety of different features within the environment, demonstrating

that the best features to choose were often dependent on the individual, or the individual's role within their organization [15]. Meanwhile, Ho, Intille et. al, demonstrated that physical activity transitions, detected using accelerometers placed on certain parts of the user's body, could be a useful measure for their interruptibility[12]. Related work by Iqbal et al. studied whether mental workload, sensed by measuring pupil dilation using a head-mounted eye-tracker, could be used to similarly infer the user's interruptibility [16].

More recent work by Horvitz et al. demonstrated that interruptibility at the desktop could be inferred from a combination of desktop application activity traces, and environmental sensors[13]. While Horvitz' work continued previous inquiries in interruptibility, his work explicitly discriminated among *types of interruptions*, and *user attentional states*. He proposed estimating the *Expected Cost of Interruption* (ECI) at a particular moment due to a particular type of disruption, D_i as follows:

$$ECI(E, D_i) = \sum_j p(I_j|E)u(D_i, I_j)$$

Where E is a vector-valued variable that represents the observed state of the user, and I_j is either a value for "attentional state" or "interruptability".

Despite this progress in this area, however, high error rates have demonstrated that 'interruptability' is a difficult variable to measure. Moreover, most of these prototype systems for filtering interruptions were poorly received by users [14]. Hudson et Ho., believed that this was most likely due to the fact that these early attempts failed to model many of the subtle aspects of what made some interrupts important to take immediately and others merely distracting [14]. In particular, Mark's interruptability study revealed that interruptions were most unwelcome when they occurred *out of sphere*, meaning that they did not pertain to the user's current task [21]. Meanwhile, interruptions that pertained to the current task often were important to take immediately, as they often advanced the task's completion. Similar results were reported by O'Conaill et. al, where it was discovered that approximately 64 percent of interruptions observed benefitted the recipient in some way [25]. Bellotti reported that whether or not to take an interruption was often influenced by social pressures, i.e., who the interruption was from [1]. Thus, deciding whether or not to take an interruption involves weighing several factors, including the interruption's relevance to the current task, the source of the interruption, the state of other tasks in the queue, as well as the user's state-of-mind, such as their workload and stress level. According to Hudson, even for professional knowledge workers such as high-level managers, deducing whether or not to take an interrupt requires a "complex tension between wanting to avoid interruption and appreciating its usefulness." [14].

We would like to propose that if the models of both the user's state and the type of disturbance were elaborated and fitted in a learning framework, the system would be able to better identify whether a person would be willing to accept an interruption. First, it would be possible to capture task-relevance relationships between interruptions and the user's current task. Second, a learning

framework could adapt to individual variations on how people like to manage their interruptions. This effectively involves a generalization of Horvitz's technique; instead of distinguishing among a very small set of classes of disturbances, ("real-time telephone", "audible" and "visual"), and interruptability states ("low", "medium", and "high"), the definitions of D_i and I_j defined earlier could be expanded to incorporate details of the user's context as well as the interrupting event. While ideally, I_j would also incorporate features identified from the earlier studies by Fogarty et al., Ho et al., and Iqbal et al., we will focus on the effect of adding the user's task context to I_j, while enriching D_j to model more aspects of the interruption. In particular, we plan to try incorporating information regarding the interruption's source, contents, and inferred relevance to the user's tasks. Based on Mark's conclusions, we believe that this task-based approach will significantly improve the system's ability to predict whether the user will be receptive to a particular interrupt.

4.2.2 Handling different types of interrupts

For simplicity, we will initially consider only two types of interruptions: instant messages and e-mail. These two types of interruptions are different due to the former being synchronous, meaning requiring both communcating parties to be participating simultaneously, while the latter is asynchronous. Considering both of these types of interruptions will allow us to generalize the system to other synchronous and asynchronous communications media. For synchronous media, any system that mediates incoming messages must make an immediate decision about whether or not to allow the message through to the user, and should inform the sender about its action. For asynchronous media, on the other hand, timing is less critical, and therefore, the system has more flexibility with regard to when and how to deliver the message to the user.

4.2.3 Identifying interrupt-task relationship

The greatest technical challenge with task-based interrupt handling is identifying, for any given interruption, whether it pertains to any of the user's known tasks. This problem could be treated as the general problem of identifying the interrupting topic and matching this topic with existing tasks. Alternatively, we could approach this problem in a similar fashion to the action-task pairing technique discussed earlier, by trying to identify characteristics of interruptions that indicate their relevance to particular tasks. This latter approach, which will likely require supervision, could employ any of the methods discussed earlier such as, the perceptron learning technique.

4.3 Redesigning the User Interface: Time-management tools that incorporate proactivity

As Blandford et. al and Bellotti's studies showed, most people use an ensemble of separate tools and applications, some digital and others paper-based. It

seems that the reason that such a variety is necessary is that no single tool available today has all of the features most people need. Most digital tools provide independent, highly structured views of tasks, such as a calendar, to-do list and address book, which allows users to set up reminders and easily organize large volumes of names and addresses. However, inputting entries into these highly restrictive structured fields often requires more time and are more tedious than sketching them. Also, by being structured and organized under a predefined schema, these systems prevent the user from freely spatially organizing things in the ways that they like. Therefore, many people keep paper-based lists readily available, which provide more spontaneous input, and representational/organizational flexibility.

A second reason that the user interface designs for today's popular time management tools have to be modified for our use is that the techniques employed in ATLIM to enable the system to take proactive action will very likely make occasional errors. Unless the user interface is designed to ensure that such errors are handled in a *fail-soft* manner, they could cause the user considerable disadvantage. Furthermore, even when the system acts correctly, if the user interface does not make it clear to the user *why* the system took a particular action, the user may not understand what the system is doing, and may grow to mistrust the system in the long run. As these two observations are critical for adoption of proactive systems, we believe that these concerns should be directly addressed from the ground-up in the design of the user interface.

As an example of a design philosophy that addresses these issues, Hudson et. al, ascribe to using inference to promote *social translucence* among workers and their collaborators [14]:

> We believe the notion of socially translucent systems [10] can provide one way to approach this challenge. Creating awareness and accountability through making behavior more visible – can allow social mechanisms to play more effective roles in technology-mediated interruptions. For example, by making information such as the current activity, location, historical patterns of activity, etc. visible in a way that does not require vigilance and active maintenance on the part of the recipient, potential interrupters can make better-informed decisions about whether to interrupt. [While even these mechanisms are not infallible], when they do fail, it tends to result in negotiation – that is, further social interaction can be used to repair or gloss over problems. On the other hand, when computer filtering fails, only anger and sometimes a sense of helplessness results. Information and attention are complex social processes that would seem to require social solutions.

While their philosophy is designed around *social processes*, such as managing interruptions from social communications channels, does it also apply to other proactive aspects of the system, such as task identification, for which action is less social (i.e, among multiple users) as it is directed towards the individual user?

If we treat the algorithm performing the task identification as a social actor, we can apply the original description of "social translucence" to system translucence in general. It makes sense to strive to make it possible for the user to be aware of the "activity, and historical patterns of activity" of this "actor", allowing the user to easily determine when it was important to intervene and correct its activities. Indeed, the notion of such an algorithm serving as a social actor could eventually become increasingly important as the mechanisms therein become increasingly capable and complex. We could imagine that some day, the user might be able to "converse" with the algorithm such as to easily express his or her desires, directly explain its behavior, or engage in negotiations. But for the moment, the social actor metaphor can be used to just improve the usability of the system. The second important aspect for such proactivity will be to ensure that the potentially erroneous actions taken by the system are to not cause harm to the user and are entirely reversible.

4.3.1 Design criteria for the user interface

Based on recommendations made by Bellotti regarding designing a better task list manager (Sec. 2.1), Marks' observations regarding automatic interrupt filtering [21], and Norman's recommendations for agent-based user interfaces [23], we will evaluate potential ATLIM designs with regard to the following heuristics:

1. *Quick, simple and flexible task entry;*
2. *Freedom for users to freely re-arrange, move, re-organize tasks visually as they see fit;*
3. *Easily accessible, constantly visible;*
4. *Incorporates ATLIM inference algorithms in a way that benefits the user, yet minimizes the risk posed by incorrect inference;* (e.g., exhibits *soft-failure*, such as when applied to improving *social translucence*.)
5. *Makes visible the following:*
   (a) action (i.e., what the system did)
   (b) accountability (i.e., why the system behaved as it did),
   (c) a means of recovery (i.e., how to undo its action),
   (d) a means of prediction/prevention (i.e., what to expect from, and account for and correct its actions in the future)
6. *Allows interactive training of the system in a way that minimizes disruption to the user, but allows for acquisition of valid training data.*

Ultimately, however, the final decision of the suitability of a user interface will be how well it is received by users, and what effect the interface has on the way the user works.

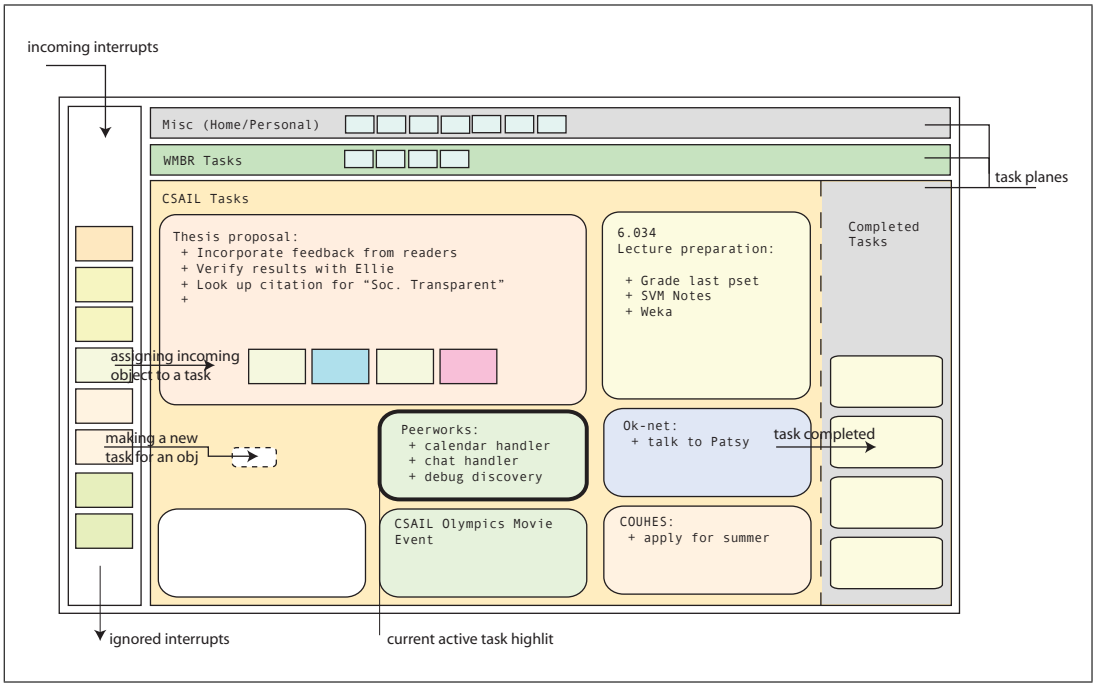

Figure 5. *Illustration of Task Matrix UI scheme for ATLIM:* In the matrix view, the task descriptions are placed in cells, which may be organized as the user sees fit at any location within the matrix. *Task planes* allow the user to organize relevant tasks by major theme, as well as by color. The system varies degree of saturation of the cells of items according to how much attention the tasks have received during the day.

4.3.2 Task Matrix: a user interface design for ATLIM

With these design criteria in mind, we wish to design a UI for an integrated to-do list, calendaring system, and interrupt manager that can use our algorithms in a way helps the user effectively. One preliminary design is the *task matrix*, visible in figure **??**.

The design of the task matrix reflects the need for both structure and flexibility, and for the user to maintain a constant ambient awareness of what the system is doing. The main task matrix consists of a grid of *task cells*, flanked on the left by the *instream* and the right by the *completion bin*. Each cell contains one logical "task" including its complete description, and optionally a decomposition into a list of sub-tasks, each of which can be individually prioritized and re-arranged. Users may rearrange task cells themselves, relocate them to a different *task plane* (high-level categories for sets of tasks), re-label and create additional task-planes as necessary. The system does not impose any formal task description schema on the user; instead, the user is provided with a tool with which they can add optional annotations such as "Deadline" and "Intended recipient". The *instream* consists all of the user's incoming messages, such as e-

mail, voice-mail and instant messages. Items from the instream can be dragged between the cells to create a new task, or onto an existing cell to associate the item with that task. Users may drag completed atoms from the task matrix onto the completion list.

The task matrix UI interacts with the ATLIM algorithms in several ways. First, as the user is working, the ATLIM highlights (in bold) the task it thinks that the user is working on. If this is incorrect, the user can correct the system by tapping on the actual task that they are working on. This example is fed into the learning algorithms to improve its performance in the future. Second, the Interrupt Manager component of ATLIM automatically takes items from the instream and attempts to associate them with the appropriate task cell. If the system makes an incorrect association, this may also be manually correct (at the user's convenience) by simply dragging the item off the incorrect cell and into the correct cell. Similarly, this reinforcement is provided to the interrupt manager to correct its behavior for future instances.

The task matrix attempts to remind the user of the distribution of their attention across their tasks increasingly coloring items that have gotten attention, while letting those items that have been neglected go pale. If the user wishes to see an explicit timeline of their work practices, this will be available in an alternate view. Similarly, a traditional calendar-based view is available to display those items with an explicit deadline annotation.

The illustrated version of the task matrix interface is intended to occupy either a portion or the entirety of one of the user's secondary displays. With multiple displays becoming increasingly common for users' desktops work environments, moving the task matrix to a secondary, (presumably smaller) display ensures that the task matrix will interfere minimally with the user's workflow, and that the user can maintain a constant peripheral (visual) awareness of the display without having to explicitly move windows out of the way. Ideally, this secondary display would also be touch-screen enabled, to make it easy for the user to interact with the system without having it interfere with the user's mousing activities.

5 Plan for Investigation

This section proposes a two-year plan for investigation of designing an automatic task-list and interrupt manager that achieves the goals outlined in section 3, by seeking solutions to the problems identified in section 4. The plan is summarized below, divided into stages:

1. *Initial data collection (6 months)* - The first step will be to build infrastructure required for collecting user data. This will be done in two sub-stages; first, infrastructure will be built for capturing action traces from users' interactions with their major desktop applications, as described in section 4.1.6. After this is complete, we will assemble a group of volunteers to run this software for collecting an initial base of action traces. The second

sub-phase will be to design infrastructure for facilitating tagging of action traces with relevant tasks. This will require obtaining the user's task-list. We will initially obtain this information from an existing calendaring tool, such as Microsoft Outlook or iCal. The tagging infrastructure should make it easy for users to either indicate what they are working on as they work on the task, or to perform tagging at the end of the day. Once such tagging is possible, we will attempt to elicit another group of volunteers to use the system.

2. *Evaluation of task-identification methods (3 months)* - In this phase, we will compare the various methods outlined in section 4 using data captured in Phase 1. The objective will be to find which combination of methods and feature representations yield the best performance, defined under the criteria described in 4.1.8.

3. *Improved data collection (2 months)* - Based on the observations in Phase 2, we will optimize the feature vectors for better performance, and modify the data collection mechanisms designed in Phase 1 to support these improved representations.

4. *Automatic Task List Manager user interface implementation (2 months)* - This stage will seek to prototype a graphical task-list manager application around the methods and representations identified in the previous sections. This user interface will reveal the task identification algorithm's ability to track the user's active tasks, and will support interactive training and tagging. This process will involve paper-prototyping, and informal design evaluations with lab members to identify usability issues with the system.

5. *Evaluation of interrupt-task correspondence methods (3 months)* - This phase will be devoted to investigating the challenge outlined in section 4.2.3, for devising a suitable algorithm that can identify how interrupts pertain to users' tasks.

6. *Interrupt management integration into ATLIM (2 months)* - This phase will take the methods devised for interrupt management developed in phase 5, and integrate them into the ATLIM UI implemented in Phase 4. This will involve interfacing with e-mail and instant messenger systems, so that new e-mails and messages can be intercepted. Also, this will involve developing a method by which users can easily specify associations between tasks and interruptions for interactive training, as well as an exploration of interrupt management policies.

7. *Iterative user evaluation (3 months)* - This phase will be devoted to collecting user impressions from the combined task list and interrupt manager. Users' impressions will be collected twice. The first set of evaluations will seek to identify critical UI design flaws. After these have been corrected, a

second set of evaluations (conducted over a 1-month period) will be performed to gain insight regarding the system's suitability for its purpose. Users will be asked to compare the tool with existing desktop time management/calendaring tools in an attempt to discern whether people are satisfied with the proactive support provided by the new algorithms.

8. *Wrap up and write-up (3 months)* - This time is allocated primarily to writing up results and conclusions.

6 Summary

This proposal presents a plan for investigating the feasibility of applying recognition methods from AI to building tools that can more actively help people keep track of their time and attentional resources. This project is an effort to pursue the notion of "calm computing" proposed by Mark Weiser in the 1990s. [30] It envisions making computers more useful by enabling them to better "understand" users by "watching" them, and automatically interpreting and learning from their actions.

The proposed investigation focuses on methods for achieving two capabilities: first, autonomously monitoring users' actions and deducing what tasks they are working on, and second, monitoring incoming interruptions and automatically identifying how to handle them. The former will enable software on the desktop to assist people's task management schemes in a number of ways, including automatically keeping track of when and how long a task was performed, organizing resources used for a task, and predicting how long new tasks will take. The latter will enable software to act as a personal receptionist, intercepting incoming interruptions and deducing whether they should be queued or passed on. This determination will be made using the user's task context, instead of any absolute measure of "interruptability". The result will more closely models user's actual interruption preferences

To test these methods, determine their suitability, and gain initial impressions from users regarding how they might change their daily work patterns, these devised methods will be incorporated into a prototype Automatic Task List Interrupt Manager application. This application, in addition to providing basic to-do list and calendaring capabilities, will be able to proactively assist users based upon knowledge of their task context in the above mentioned ways. Feedback gained from this experience, along with performance results from the analysis algorithms, will be used to make recommendations for the design of future time and task-management tools.

References

[1] Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G. Bobrow, and Nicolas Ducheneaut. What a to-do: studies of task management

towards the design of a personal task list manager. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 735–742, New York, NY, USA, 2004. ACM Press.

[2] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 345–352, New York, NY, USA, 2003. ACM Press.

[3] A. E. Blandford and T. R. G. Green. Group and individual time management tools: What you get is not what you need. *Personal Ubiquitous Comput.*, 5(4):213–230, 2001.

[4] Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[5] Michael Collins. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *IWPT*. Tsinghua University Press, 2001.

[6] Michael Collins. Lecture notes on global linear models for 6.864: Advanced natural language processing, 2005. http://people.csail.mit.edu/regina/6864/slides/l20.pdf.

[7] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. A diary study of task switching and interruptions. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, New York, NY, USA, 2004. ACM Press.

[8] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[9] Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82, New York, NY, USA, 2005. ACM Press.

[10] Thomas Erickson and Wendy A. Kellogg. Social translucence: an approach to designing systems that support social processes. *ACM Trans. Comput.-Hum. Interact.*, 7(1):59–83, 2000.

[11] Werner Geyer, Jorgen Vogel, Li-Te Cheng, and Michael Muller. Supporting activity-centric collaboration through peer-to-peer shared objects. In *GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work*, pages 115–124, New York, NY, USA, 2003. ACM Press.

[12] Joyce Ho and Stephen S. Intille. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 909–918, New York, NY, USA, 2005. ACM Press.

[13] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, pages 20–27, New York, NY, USA, 2003. ACM Press.

[14] James M. Hudson, Jim Christensen, Wendy A. Kellogg, and Thomas Erickson. "I'd be overwhelmed, but it's just one more thing to do": availability and interruption in research management. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 97–104, New York, NY, USA, 2002. ACM Press.

[15] Scott Hudson, James Fogarty, Christopher Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny Lee, and Jie Yang. Predicting human interruptibility with sensors: a wizard of oz feasibility study. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 257–264, New York, NY, USA, 2003. ACM Press.

[16] Shamsi T. Iqbal and Brian P. Bailey. Investigating the effectiveness of mental workload as a predictor of opportune moments for interruption. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1489–1492, New York, NY, USA, 2005. ACM Press.

[17] Victor Kaptelinin. UMEA: translating interaction histories into project contexts. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 353–360, New York, NY, USA, 2003. ACM Press.

[18] D. Keysers and H. Ney. Linear discriminant analysis and discriminative log-linear modeling. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 1, pages 156–159. IEEE, August 2004.

[19] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[20] Blair MacIntyre, Elizabeth D. Mynatt, Stephen Voida, Klaus M. Hansen, Joe Tullio, and Gregory M. Corso. Support for multitasking and background awareness using interactive peripheral displays. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 41–50, New York, NY, USA, 2001. ACM Press.

[21] Gloria Mark, Victor M. Gonzalez, and Justin Harris. No task left behind?: examining the nature of fragmented work. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–330, New York, NY, USA, 2005. ACM Press.

[22] Nicholas Kushmerick Mark Dredze, Tessa Lau. Automatically classifying emails into activities. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*. ACM Press, 2006.

[23] Donald A. Norman. How might people interact with agents. *Commun. ACM*, 37(7):68–71, 1994.

[24] Chintan Thakkar Nuria Oliver, Greg Smith and Arun C. Surendran. Swish:semantic analysis of window titles and switching history. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*. ACM Press, 2006.

[25] Brid O'Conaill and David Frohlich. Timespace in the workplace: dealing with interruptions. In *CHI '95: Conference companion on Human factors in computing systems*, pages 262–263, New York, NY, USA, 1995. ACM Press.

[26] George Robertson, Eric Horvitz, Mary Czerwinski, Patrick Baudisch, Dugald Ralph Hutchings, Brian Meyers, Daniel Robbins, and Greg Smith. Scalable fabric: flexible task management. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 85–89, New York, NY, USA, 2004. ACM Press.

[27] Ben Shneiderman. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 33–39, New York, NY, USA, 1997. ACM Press.

[28] G. Smith, P. Baudisch, G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, and D. Andrews. Groupbar: The taskbar evolved, 2003.

[29] Clive Thompson. Meet the life hackers. *The New York Times Magazine*, October 2005.

[30] Mark Weiser and John Seely Brown. The coming age of calm technology. Technical report, Xerox PARC, 1996.