TaskTracer: A Desktop Environment to Support Multi-tasking Knowledge Workers

Anton N. Dragunov, Thomas G. Dietterich,
Kevin Johnsrude, Matthew McLaughlin,
Lida Li, Jonathan L. Herlocker
Oregon State University
Department of Computer Science
102 Dearborn Hall
Corvallis, OR 97331, U.S.A.
{anton, tgd, johnsrud, mclaughm, lili, herlock}@cs.orst.edu

ABSTRACT
This paper reports on TaskTracer — a software system being designed to help highly multitasking knowledge workers rapidly locate, discover, and reuse past processes they used to successfully complete tasks. The system monitors users' interaction with a computer, collects detailed records of users' activities and resources accessed, associates (automatically or with users' assistance) each interaction event with a particular task, enables users to access records of past activities and quickly restore task contexts. We present a novel Publisher-Subscriber architecture for collecting and processing users' activity data, describe several different user interfaces tried with TaskTracer, and discuss the possibility of applying machine learning techniques to recognize/predict users' tasks.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces — *User-centered design, Prototyping.*

General Terms
Management, Design, Human Factors.

Keywords
Knowledge management, multitasking, activity monitoring, machine learning, user interface.

1. INTRODUCTION
Knowledge workers spend the majority of their working hours processing and manipulating information. They take information as input and produce information as output. The information may be encoded in many different formats: documents, software code, web pages, email messages, phone conversations. An important characteristic of knowledge workers is that their work is cognitively intensive and requires focus, concentration, and memory. Another characteristic is that they must process considerable quantities of information in order to get their job done.

© ACM, 2004. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in the Proceedings of the 10th international conference on Intelligent user interfaces (San Diego, California, U.S.A., January 9-12, 2005), http://doi.acm.org/10.1145/1040830.1040855

Many of today's professionals are knowledge workers: professors, managers, software developers, lawyers. Many of these professionals have multiple tasks in progress concurrently, although they are generally only working on a single task at any instant in time. The combination of cognitively intensive processing, considerable quantities of information, and multitasking make knowledge work extremely challenging [6, 11]. We consider how we can design intelligent user interfaces (UI) to make knowledge work less challenging and more productive. We focus on knowledge workers who interface with information primarily through a desktop software interface.

This paper reports on *TaskTracer*, our current attempt at Oregon State University to create a software system to assist knowledge workers in their daily routines. Our software focuses on two areas where we believe that intelligent software interfaces can have a substantial effect on the productivity of knowledge workers: interruption recovery and knowledge reuse.

By definition, highly multitasking people face continual interruptions as they switch between ongoing tasks [7, 11, 18]. Given that knowledge workers are involved in nontrivial analysis and deal with large amounts of information, recovering from interruptions often has a significant overhead cost. This overhead may be cognitive: workers may have to remember exactly where they were in a chain of logic, or why they decided to take their most recent action on a task. The overhead may also just lie the in manual interaction needed to locate and access the necessary resources (e.g., documents and/or software tools). If we can reduce the overhead involved during interruption recovery, we should be able to improve the productivity of knowledge workers.

Knowledge workers often manage the complexity of their work through extensive knowledge reuse. One common method of this reuse is what we call *templating*. Templating occurs both with processes and with artifacts (most commonly, documents) and is possible because knowledge workers must often repeat very similar processes. For example, researchers must write grant proposals every year. Knowledge workers use past processes as templates for their current work. Researchers reuse the process that successfully gained them the last grant proposal as a template for a new proposal. This eliminates the need to re-derive (cognitively or manually) the processes needed to complete a task. Furthermore, it allows iterative improvement of processes over time. Finally, processes known to succeed in the past may have high likelihood of succeeding in the future. We seek to design software that helps people to rapidly locate, discover, and reuse past processes they used to successfully complete tasks.

In some interaction scenarios, it seems less useful to recall an entire past process and rather recall individual information re-

sources that are generally useful for particular types of tasks. For example, the process of hiring graduate students is significantly different from hiring permanent staff, yet the human resources (HR) web site at Oregon State University is very useful for both. A professor may have never hired a permanent staff member before, but he/she can return to the HR page to seek guidance.

With TaskTracer, we are seeking to address these issues of interruption recovery and knowledge reuse using a combination of ubiquitous (across software applications) data collection, machine learning, information visualization, and recommendation UI's.

2. RELATED WORK

The idea of building an environment that enables knowledge workers to manage multiple concurrent activities and use knowledge of those activities to improve productivity has been around for many years. All the approaches suggested so far, in one way or another, are based on the premise that knowledge workers organize their work into discrete units, usually called *tasks*. Tasks define virtual workspaces that comprise information resources (usually documents and tools for their processing) necessary to accomplish the goal associated with the task. Some systems allow (in a sense) "physical" separation of tasks requiring users to create a project-specific folder, or set up a virtual desktop for each particular task (e.g., [5, 20]). Other systems work at more abstract level by organizing task-specific workspaces using "filters" applied to communication *threads* (as in TaskMaster [3]), *streams* or *networks* of documents (Lifestreams [10] and Presto [8] respectively).

To be of assistance to a user, an agent (whether it is a computer system or a human associate) must "understand" what the user is currently doing. It has long been believed that a computer assistant can infer information about users' tasks and goals by analyzing the context in which the user performs one action or another [1, 14]. In some information retrieval systems, such as Watson [4], the currently opened document is used for context to improve the accuracy of search results. However, the user's task context usually includes not only the current document, but also other documents, as well as contacts, electronic messages, and many other items, which might not be present in the workspace at the moment, but which were used in the past or are scheduled for use in the future.

In task-centered communication systems, recording the *history* of the task (and/or the virtual workspace associated with it) is addressed by creating contexts out of threads of messages sent and received. Communication threads, such as those in Bellotti's TaskMaster [3], or temporal grids as in Gwizdka's TaskView [12], seem to quite accurately reflect history of the project (or task) and describe a variety of resources used in it: contacts, electronic messages, task-related documents, etc. One significant drawback of such systems, however, is that important information that does not go through the communication channel may not be recorded at all. For example, relevant web links may appear in e-mail messages, but the web-browsing history that led to those links is likely to be lost forever. A web link in a message leaves a track about a particular web page, but not about the experience (and thus the certain amount of *knowledge*) obtained during the course of accessing the page. Extra information "around" a document, web-link, or contact may provide many important details for the task context.

In addition to the records about resources used in a task, it also seems reasonable to record user's actions performed on those resources. The rationale behind this is that to have the correct comprehension of the task context for some resources we must consider in which way and for what reason they were accessed. For instance, the same document (say, a text file) may be opened for two completely different purposes: 1) for reading and 2) for authoring. The approach by Kaptelinin in his UMEA (User-Monitoring Environment for Activities) system [15] addresses this issue by aiming at recording as much information as possible about users' activities when they interact with computers. Activity records are obtained via monitoring the computer file system, input devices, and running applications.

The dilemma associated with this approach is that collecting the system events for activity records at the very fine level of granularity (which is desired) may create overhead and huge amounts of data that may be challenging to draw inferences from [13], or may not be useful at all in some tasks. Reducing the amount (and the variety) of the data collected lowers the requirements for systems hardware/storage and reduces the complexity of inference modules, but it may leave some important data unrecorded and unprocessed. Thus, the UI event monitoring system should balance the granularity of data collection and the necessary level of inference.

It seems beneficial to approach this problem by providing users with the possibility to tailor the data collectors for each particular task. For example, some tasks, such as typing or drawing, may require information about every single key stroke and mouse move, whereas other tasks may require only very high-level events such as opening an application (e.g., running an antivirus program to scan a hard drive). Ideally, the data collectors should be tailorable for every particular application, so that users can decide what data is important to gather from that application.

An attempt to design such a system was reported recently by Fenstermacher and Ginsburg [9]. In their project, called POKER (Process-Oriented Knowledge Delivery), the event monitoring is initiated by launching a shell application written in Python that in turn launches Microsoft applications in the form of COM objects. The program then monitors the events exposed by these COM objects. However, such invocation can be inconvenient, since it requires the user to remember to start the shell application prior to launching any monitored applications. This architecture restricts the number of events that can be monitored to those that are exposed by the COM interface.

3. THE TASKTRACER

TaskTracer adopts the idea that knowledge workers organize their work into tasks. Associated with each task is a process to complete the task and a set of information resources (documents, electronic messages, contacts, etc.) and tools (computer applications, phone line) employed to access and manipulate these resources.

3.1 Task Profiles

Through an extensive data-collection framework, TaskTracer collects detailed observations of user interactions in the common productivity applications used in knowledge work: email, word processing, spreadsheets, and Internet browsers. At the initial stage of data collection, users manually specify what tasks they are doing, so that each action of the user (UI event) will be tagged

with a particular task identifier. We believe that with enough data we can learn to reliably predict the users' current task and task switches, and thus we can create complete and detailed records of what has been done on every tasks (past and present). We call such a record of a task a *task profile*.

Our goal is to leverage task profiles to support interruption recovery and knowledge reuse. When users return to a task they have previously interrupted, TaskTracer can restore all the applications that were in the process of being used for that task, open the documents that were being consulted, and indicate within those documents what elements had been recently changed at the time of the interruption. Furthermore, TaskTracer's knowledge of the current task and collection of all past task profiles allows it to customize the user interface to most efficiently support that current task. For example, file open dialogs can default to folders associated with the current task, files most likely to be needed can be placed on a quick-start bar, and applications likely to be needed can be preloaded into memory.

3.2 UI Event Monitoring

While following the same path as UMEA and POKER in seeking to gather as much information about user's activities as possible, TaskTracer does the data collection in a slightly different way. TaskTracer too uses Microsoft's COM components, but everything is done in the Microsoft .NET Framework. Python, used in POKER, is an excellent approach for rapid application development, but Microsoft .NET provides greater and finer-grained monitoring capabilities, albeit with more initial development effort.

TaskTracer monitors Microsoft Office, Visual Studio and Internet Explorer applications by installing .NET COM addin objects with the Extensibility and IObjectWithSite interfaces. These addins monitor Microsoft applications as soon as they are launched and are intimately bound to the applications. No user intervention or shell application is required once TaskTracer is installed. The addins also access the richer event sets of the Microsoft Office internal Visual Basic for Applications (VBA) compiler. Using VBA, TaskTracer can monitor a richer event set by working around Microsoft Office COM limitations.

TaskTracer also monitors Windows at the operating system level with such events as window focus, clipboard and file creation events. To monitor applications and the operating system TaskTracer uses components written in C#, C++, and VB.NET.

3.3 The Publisher-Subscriber Architecture

TaskTracer separates the user interface and data analysis components from the event collection components by using a *Publisher-Subscriber Architecture* (Figure 1).

The Publisher collects data about the user's activities and disseminates this event data to one or more Subscribers. Each Subscriber can process the event data from the Publisher in a different way. Some Subscribers not only receive EventMessages from the Subscriber Port but also send EventMessages to the Listener Port. For example, TaskExplorer (a UI for task switching described later in the "User Interface" section), sends TaskBegin messages to the Listener Port every time the user selects a new task.

Events are collected from

- Task.Connect, a COM addin attached to MS Office applications and that can also communicate with the Visual Basic for Applications (VBA) macros within MS Office applications.
- A Windows CBT hook which is attached to the Publisher.
- .NET FileSystemWatcher class which is attached to the Publisher.
- A hook to the Windows Clipboard chain which is attached to the Publisher.
- A hook to a phone modem which collects Caller ID and speech-to-text information.

Task.Connect collects an MS Office event and assembles the event into an EventMessage (thin arrows on Figure 1) that is sent via TCP to the Publisher. The Publisher receives an EventMessage, stores it in a database and sends it, via TCP, to Subscriber applications for further processing. Depending on the research question and goal pursued, each Subscriber can process the EventMessages received from Publisher in a different way.

All EventMessages are stored in the Publisher's database in raw form so that researchers can analyze the history of user events. Since the learning models are still being developed, any data

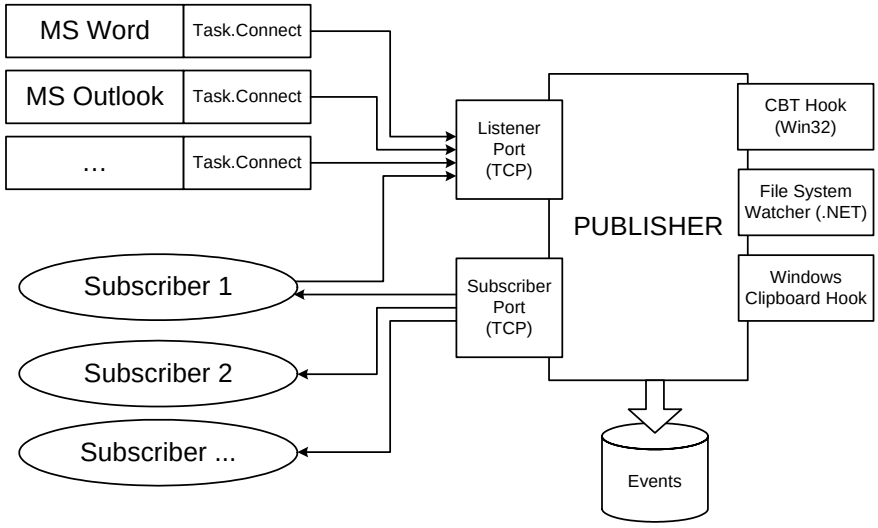

Figure 1. TaskTracer Publisher-Subscriber Architecture

analysis that changes the data is premature. A variety of learning models can be tested on identical data sets. We are currently researching learning models based on the event data for predicting the current task of the user and for detecting when the user has changed tasks.

An EventMessage contains

- *Type*: Event type. For example, TaskTracer captures window focus, file open, file save, web page navigation, text selection, and many other events on both the applications and the operating system levels.
- *Window ID*: Window handle for windows, zero otherwise.
- *Listener Version*: Changes every time we change or add to the EventMessages the Listener can send and process. This allows backward compatibility as we change our data capture.
- *Listener ID*, the source of the EventMessage: MS Office programs, file system hooks, user, clipboard, phone, etc.
- *Body Type*, *Body*: Body Type and Body in XML format.
- *Time*: Time the event fired.

EventMessages are events collected from Microsoft Office 2003, Microsoft Visual .NET, the Windows XP operating system and phone calls. Currently TaskTracer collects events from Microsoft Word, Excel PowerPoint, Outlook and Internet Explorer with future plans to collect events from Microsoft Access, FrontPage, Project, Publisher, MSN Messenger and Visio. Work is underway to collect data from non-Microsoft applications such as Acrobat Reader and Mozilla/Netscape.

TaskTracer currently collects file pathnames for file create, change, open, print and save. Text selection and copy-paste events are also collected. Work is underway to collect file data such as full documents in XML format, Mail Merge, Excel Calculations, Excel Pivot Tables and anything exposed by the MS Office OLE Automation objects or by Microsoft VBA.

TaskTracer also collects Windows focus, web navigation, phone call, clipboard and email events. We are also working on constructing generalized file open event capture for all Windows applications. Phone call data collection uses Caller Id to collect names and phone numbers of callers. In addition, speech-to-text software collects the user's — but not the caller's — phone speech.

The Publisher-Subscriber architecture of TaskTracer has several powerful advantages over a monolithic approach. These advantages include the following:

- Data collection is separated from data analysis and the user interface.
- No prejudgments are made about the data schema of the data collected.
- Multiple researchers can work on multiple projects with multiple data schemas that subscribe to the Publisher without interfering with each other.
- Raw event data stored by the Publisher is like a tape-recorder and can be analyzed and even replayed at a later time.

The separation of the data collection components from the analysis and presentation components allows rapid application development of new software projects by researchers. For example, to create a new UI, a researcher only needs to understand the structure of the EventMessage and the use of the TCP protocol. The ability to send EventMessages simultaneously to many Subscribers allows separate independent research projects to assemble a suite of applications that can function as a single application.

Figure 2. TaskTracer UI: Pop-up Menu.

The separation of the data collection components from the Publisher and Subscriber components has allowed the TaskTracer developers to concentrate on event capture arcana without the distraction of the other components with the result that TaskTracer can capture a wide breadth of event types with a corresponding richness of event data.

3.4 User Interface

We are currently experimenting with different user interfaces for TaskTracer.

3.4.1 Pop-up Menu

The interface is shown on Figure 2. The main UI element is a label (semi-transparent when not in focus of input) showing the title of the current task. The label is always on top of all application windows. This also serves users as a constant reminder about what they are currently working on.

Clicking on the label brings up a pop-up window containing menu-like items for quick access to the resources (documents and web-pages) associated with the current task. Also, it provides quick access to other tasks.

The color scheme for this UI may seem rather unusual (bright yellow-green text on dark-green background). This was done on purpose to make the UI be easily distinguishable from "regular" MS Windows applications. TaskTracer is supposed to be on top of every desktop activity, so the special status of this application needs to be emphasized in some way, e.g., by using an extraordinary color scheme.

It turned out that the interface had several significant drawbacks. First of all, some users found the task title on the screen to be

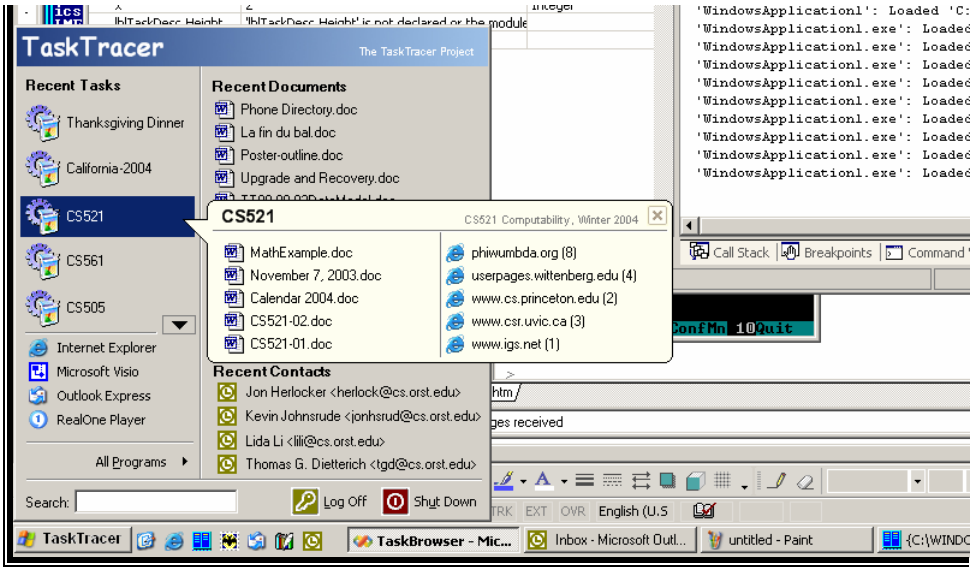

Figure 3. TaskTracer UI: Start Menu.

irritating. Although the label could be easily moved around with a mouse, there was always a chance that it would go over and thus block a control (button, input field, window frame border) which the user needed to use at that moment.

Another big issue concerned the lists of recent items (tasks, documents, web-pages). For some users, seven recent tasks and a dozen documents in the list were quite enough, but there were people whose lists of ongoing tasks and working documents were huge (hundreds), and they needed to have quick access to *all* of them. So, the lists showing only several recent items were useless for such users. They always were invoking the full lists, sorted alphabetically or otherwise, and found them more convenient to use.

3.4.2 Start Menu

In an attempt to reduce the inconvenience resulting from the introduction of extra UI elements on the screen, we considered a closer integration of the TaskTracer UI into the operating system. The main idea was to use existing user interface elements of Windows: "Do not add anything, but *replace* existing elements with ones having desired functionality." The conjecture was that native Windows interface elements were already familiar to users, so they would not need to master a new interface concept. TaskTracer would only rearrange (and in some cases add a few task-specific items into) standard menus, toolbars, etc.

Figure 3 shows a prototype interface whose basic design idea was to make the standard Start Menu of Windows work as an interface to switch tasks and access task-specific resources. The standard Start Menu of MS Windows already contains many elements (shortcuts) to access different computer resources. For example, there are already lists of recently used applications and documents. At the very least, we only need to limit those lists by the tasks they apply to and provide several new controls/shortcuts for task switching and manipulating task profiles.

The title of the current task shows up on the Start button in the task bar. The title and description of the task are shown in the menu caption (replacing the user's login name, which appears there in standard start menu of MS Windows).

The balloon on Figure 3 is a "peek and select" feature of the interface: if users do not remember in which of the recent tasks they have a particular document or web-site, they can move mouse pointer over the task names and see brief information about the task: what that task is, what documents are in there, when the task was last accessed, etc. After such "peeking" into the task, they can finally decide which task they need to switch to.

However, the Start Menu UI has the same problem as we experienced with the Pop-up Menu described earlier — limiting lists of

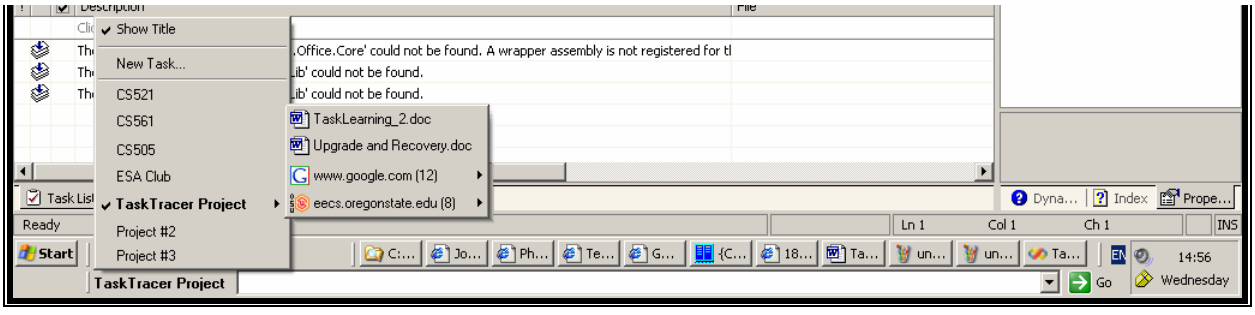

Figure 4. TaskTracer UI: Toolbar.

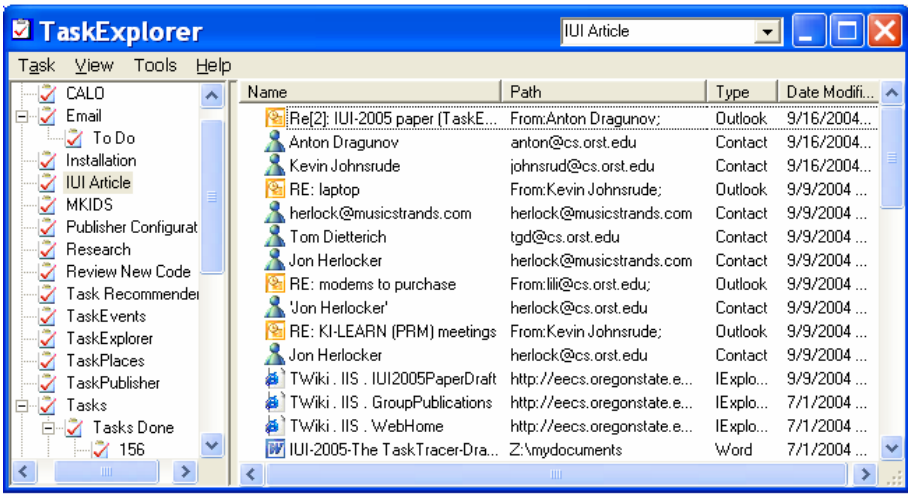

Figure 5. TaskTracer UI: TaskExplorer.

recently-used resources to a number of items which could be laid out on the screen without creating too much clutter (5–20 items in a list) was still inconvenient if users needed to operate on hundreds of tasks and/or documents.

Customizing the Start Menu also required a certain amount of "hacking", since there were no standard tools in Windows to change its appearance and functionality.

3.4.3 Toolbar

Another standard control we tried to implement for TaskTracer was a *toolbar*. Figure 4 shows this prototype control with a task-switching menu invoked by clicking on the toolbar caption which displays the title of the current task.

The toolbar is placed in the Windows taskbar, which can be customized to be on the screen at all times or auto-hide itself when not used — a feature appreciated by users who find extra labels on the screen irritating.

The functionality of the toolbar is similar to that of the standard Windows address bar. In addition it supports task-related stuff: switching between tasks, locating and opening documents/websites, etc. To locate and use a resource, users can simply start typing the name of the resource in the input box. The system autocompletes users' input suggesting a list of filenames, tasks or applications to choose from. This feature is for users who prefer to use the keyboard instead of the mouse. For such users, launching an application or opening a document by typing a file name in the TaskTracer toolbar can be done much faster than locating the resource using a browser (e.g., Windows Explorer).

3.4.4 TaskExplorer

TaskExplorer is the latest experimental UI for TaskTracer. It uses the Windows Explorer as a metaphor. TaskExplorer is both a Subscriber and a Listener. TaskExplorer subscribes to the Publisher to receive EventMessages from Listeners and also sends an EventMessage to the Publisher whenever a new task is selected. TaskExplorer deliberately uses a very small subset of the information collected by the Listeners to explore the usefulness of a minimal interface.

The TaskExplorer window is shown in Figure 5. The left pane is a tree view of the user's tasks and subtasks. The right pane is the collection of resources associated with a particular task. The widths of these columns are user-configurable. TaskExplorer adds resources based on the EventMessages received from the Publisher. If a resource already exists for a particular task, then the Date Modified field is updated. Resources can be sorted by Name, Path, Type or Date Modified in ascending or descending order. If the user clicks on a resource, the resource will be opened by the application associated with it.

The combo box in the TaskExplorer title bar shows which task the user is on. This title bar combo box will move to whatever window is in focus except for topmost windows and dialog boxes. This simple design solution unexpectedly received positive feedback from the current users of TaskTracer, since it allows the user to see the current task and quickly access other tasks, and yet the UI element is not blocking any working area on the screen (the region of the window title bar where the combo box is placed usually does not contain any useful information). The user switches between tasks either by selecting a task name from the combo box or by clicking on it in the TaskExplorer tree view. The user selects tasks from the title bar combo box either by using the pull-down menu or by typing the first few letters of the task name.

To create a new task, the user clicks on "Task | New..." menu or enters a new task in the title bar combo box. The user can arrange tasks hierarchically by dragging and dropping tasks underneath other tasks within the task tree view. Documents can be moved from one task to another by dragging and dropping.

Keyboard shortcuts are available for all TaskExplorer elements if the user prefers not to use the mouse.

3.4.5 Other UIs

TaskTracer can be considered as a kind of a virtual desktop manager [5]. When users switch tasks in a virtual desktop, all applications from the current task should be hidden from view and, if necessary, hibernated, and the applications from the new task should show up exactly as users left them previously. Hiding documents and applications not related to the current task can help better separate users' tasks and thus reduce the "noise" in task-tagging of the activity data collected.

Another interface concept that comes very close to the one of virtual desktop manager and that can be tried with the TaskTracer is the GroupBar suggested recently by Microsoft Research [2]. With the GroupBar, users can create groups of buttons on the task bar for different applications related to the same task. A similar feature is implemented in MS Windows XP, but it automatically groups together all windows of a single application. With user activity monitoring and event collection of TaskTracer, GroupBar can become a powerful productivity tool for multitasking users.

4. CONCLUSIONS AND FUTURE WORK

In its current version, TaskTracer is used by the members of our research group primarily as an instrument for collection of user activity data. Not yet tested in the real work environment, TaskTracer now is rather a research tool that helps us identify a variety of research questions and suggest approaches to solve them.

The big question we are trying to answer now is how we can collect the cleanest, most accurate data on what tasks exists, and what observed events belong to which tasks. Collecting data with absolutely no noise seems to be impossible, even in the laboratory environment of a small research group, where each member is interested in obtaining pure data and thus tries to be very meticulous in task specification. Users still make errors and tend to forget to indicate when they switch tasks. Therefore, support for manual post-processing of collected data (adding, deleting, or retagging of UI events) may be required.

We already have started and are going to continue investigating machine learning approaches to detect task switches and predict what the current task is (or if a new task has started). We are also going to experiment with different learning models and collaborative filtering techniques to predict what resources/tasks might be relevant to user's current task, so that we can provide users with useful recommendations and make the relevant resources be easily accessible.

As pointed out by Kaptelinin in [15], the whole idea of automatic translation of interaction histories into project contexts is very challenging to implement. If users must indicate task switching manually (as currently implemented in TaskTracer), this will create additional cognitive and physical burden for users, since they will have to 1) mentally structure their activities and 2) perform additional actions not directly related to the current goals — select tasks from lists, type in task titles and descriptions, etc. We believe that we can reduce this burden by combining machine learning with appropriate user interfaces.

Interaction with technology is full of trade-offs along several dimensions of difficulty [16]. For example, if a user wants to find something on the Web, and he/she knows that one of the virtual desktops on the computer already contains a web-browser pointed to, e.g., Google, the user is likely to switch to that desktop to do the web-search. Launching a new web-browser window and typing in the URL in the current desktop will require more time and effort than using the existing browser in another desktop which is just a click away. Even if another desktop represents an entirely different task that should not be "spoiled" with an irrelevant web-search, the user may find it more convenient at the moment to introduce a small amount of "noise" into the task-tagged event stream, than to destroy his or her current mental context by additional efforts to fight the technology.

We need to investigate how users devise and employ such strategies in order to find practical compromises necessary for workers to run our software on their desktops every day and benefit from it. User studies in the real work environment together with a thorough analysis of activity histories should also give us much of insight into how to design effective user interfaces for TaskTracer that would help streamline users' interaction based on knowledge of their past and current tasks.

Another big challenge is the tracking of users' activities that do not involve direct interaction with a computer. Technology to capture that sort of user activity already exists [17, 21].

Our support for recording and transcribing phone conversations is another step towards solving this problem — currently we are able to detect incoming calls, identify callers, and automatically switch the user to the tasks associated with the caller. However, existing speech recognition modules perform well only on outgoing speech recorded directly from the user's microphone. The caller's speech is almost impossible to recognize automatically due to relatively poor sound quality in the phone line. (The recognition accuracy is only about 20%).

5. ACKNOWLEDGEMENTS

This project was supported in part by the National Science Foundation under grant IIS-0133994 and by the Defense Advance Research Projects Agency under grant HR0011-04-1-0005.

6. REFERENCES

[1] Bannon, L., Cypher, A., Greenspan, S., Monty, M., Evaluation and analysis of users' activity organization. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, p.54–57. Boston, Massachusetts, USA. ACM Press, 1983.

[2] Baudisch, P., Smith, G., Robertson, G., Czewinski, M., Meyers, B., Robbins, D., Horvitz, E., Andrews, D., GroupBar: The TaskBar evolved. *Proceedings of OZCHI'03*, 2003.

[3] Bellotti, V., Ducheneaut, N., Howard, M., Smith, I., Taking email to task: The design and evaluation of a task management centered email tool. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp.345–352. Ft. Lauderdale, Florida, USA. ACM Press, 2003.

[4] Budzik, J., Hammond, K.J., User interactions with everyday applications as context for just-in-time information access. *Proceedings of the 5th international conference on Intelligent user interfaces*, pp.44-51, ACM Press, 2000.

[5] Card, S., D. Austin Henderson, Jr., Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, *ACM Transactions on Graphics*, vol.5(3), pp.211–243, ACM Press, 1986.

[6] Cypher, A., The structure of user's activities. In: Norman, D.A., Draper, S.W. (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates,. 1986.

[7] Czerwinski, M., Horvitz, E., Wilhite, S., A diary study of task switching and interruptions. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp.175–182. Vienna, Austria. ACM Press 2004.

[8] Dourish, P., Edwards, W.K., LaMarca, A., Salisbery, M. Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction*, vol.6(2), 1999.

[9] Fenstermacher, K.D., Ginsburg, M. A Lightweight Framework for Cross-Application User Monitoring. *IEEE Computer*, vol.35(3), pp.51–59. IEEE, Inc. 2002.

[10] Freeman, E.T., Gelernter, D., Lifestreams: A storage model for personal data. *SIGMOD Record*, vol.25(1), pp.80–86. ACM Press, 1996.

[11] González, V.M., Mark, G., "Constant, constant, multitasking craziness": Managing multiple working spheres. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp.113–120. Vienna, Austria. ACM Press, 2004.

[12] Gwizdka, J., TaskView: Design and evaluation of a task-based email interface. *Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada. IBM Press, 2002.

[13] Hilbert, D., Redmiles, D.F. Extracting Usability Information from User Interface Events. *ACM Computing Surveys*, vol. 32(4), pp.384–421, December 2000.

[14] Huff, K.E., Lesser, V.R. Knowledge-based command understanding: An example for the software development environment. *Technical Report 82-6*. Computer and Information Science, University of Massachusetts, Amherst, MA, USA. 1982.

[15] Kaptelinin, V., UMEA: Translating interaction histories into project contexts. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp.353–360, Ft. Lauderdale, Florida, USA, ACM Press, 2003.

[16] Lansdale, M. The Psychology of Personal Information Management. *Applied Ergonomics*, vol.19(1), pp.55–66, 1988.

[17] MacIntyre, B., Mynatt, E.D., Voida, S., Hansen, K.M., Tullio, J., Corso, G.M., Support for multitasking and background awareness using interactive peripheral displays. *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pp.41–50. Orlando, Florida, USA. ACM Press, 2001.

[18] McFarlane, D.C., Latorella, K.A., The Scope and importance of human interruption in human-computer interaction design. *Human-Computer Interaction*, vol.17, pp.1–61, 2002.

[19] Pederson, T. Magic Touch: A Simple Object Location Tracking System Enabling the Development of Physical-Virtual Artefacts in Office Environments. *Journal of Personal and Ubiquitous Computing*, 5, 2001.

[20] Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D., Gorokhovsky, V., The Task Gallery: A 3D window manager. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp.494–501. The Hague, The Netherlands. ACM Press, 2000.

[21] Want, R., Hopper, A. The active badge locator system. ACM *Transaction on Office Information Systems*, 10 (1), 1992