

TaskMinder: A Context- and User-Aware To-do List Management System

Brian M. Landry, Rahul Nair, Zach Pousman, Manas Tungare

Georgia Institute of Technology / GVU Center

801 Atlantic Drive

Atlanta, GA 30332 USA

{blandry, rnair, zpousman, manas} @ cc.gatech.edu

ABSTRACT

We describe the design, functionality, and early-stage evaluation of TaskMinder, a to-do list management application. TaskMinder builds in rich environmental context, including location and user activity, and uses a machine learning algorithm to compare against a user's history to produce an automatic ranking of to-do list items. The application also can suggest tasks to be accomplished given short term time constraints. We present key aspects of the TaskMinder design, discuss its implementation and evaluation, and suggest further work in the field of adaptive personal information agents.

INTRODUCTION

Users employ many different types of computer (and non-computer) technology to manage their personal to-do lists. Desktop and handheld computing platforms alone have hundreds of applications available for to-do list management, and PDA hardware and PIM software alone is a \$6.6 Billion business annually [1]. Nearly all office productivity software packages include personal information management functionality. However, to-do list management components are often simplistic [12]. Their high cost for users and low pay-off in terms of increased performance is at odds with other PIM tools, which have advanced considerably.

Task management systems are, by and large, quite rudimentary. They afford task entry, a way to view incomplete and complete tasks, and a way to mark tasks as complete. Most allow the entry of some metadata (such as due date and category) and search capability. To-do list management tools are mostly tied to calendar and email tools, since many to-do list items are triggered by email or calendar events. The tools, are, however, time consuming to manage and keep up to date [2].

The contextual richness offered by commercial to-do list managers is non-existent, as is even rudimentary learning about the user and her mode of operation. Beyond the task title itself, little more can be added. And even if metadata can be appended to a to-do item, the time required to specify a task increases as the specifying process gets more complex. Because current systems lack context, they suffer in helping a user to manage to-do items. Some of the tasks

that users have can only be accomplished at certain times of day, or in certain locations. But the applications are of no help here. Lastly, the tools make no effort to make suggestions, or be persuasive computing elements.

We propose a system that seeks to overcome the current challenges. We introduce a task management system, TaskMinder, which is dynamic, persuasive, and contextually based. Our software utilizes machine-learning algorithms to account for environmental context and user activity. We begin with related work, go on to detail the design and development of the system, and discuss a qualitative evaluation of its usability, likeability, and intelligence.

RELATED WORK

Many current tools exist for the management of to-do lists. Some are simple, like the tools in Microsoft Outlook and Lotus Notes. There are, however, some systems that have taken steps to mitigate some of the shortcomings mentioned in the introduction.

comMotion is primarily a map- and location-based to-do list application [8]. The novelty in the system is that machine learning is used to learn locations (from clusters of GPS points) instead of manually entering them. comMotion is suggested as a ubiquitous computing platform for all kinds of location-specific data, not a task management component. Another tool with a similar focus is Dey and Abowd's CybreMinder [6] that brings environmental context to reminders. Some reminders that users send to themselves can be accurately referred to as to-do list items, and the system brings unlimited contextual information via the Context Toolkit [11]. However, the system requires explicit specification of context, which increases the time it takes to put reminders into the systems.

Some studies have borne our suspicion that to-do list applications are complex and therefore underutilized. The time required to input data, is an important factor as shown with calendar data by Wong, Starner, and McGuire [13]. 66% of respondents in their study did not enter a sample appointment into their primary scheduling system (even a paper based agenda), but instead chose to use a faster

access system that was available, including writing on scratch paper, writing on skin, or memorizing. Results improved slightly for complex appointments, but were still plagued with low entry into a given user's calendar manager of choice. Peters [10] performed ethnographic studies of individual's personal information management systems and suggests that context, both spatial, temporal, and event-driven changes the way people want to access their information. "*Events or other outside forces will often cause me to change the location or label of a particular piece of information due to a change in priority or object valuation,*" said one of his informants. We seek to bring statistical modeling techniques to bear on this problem.

In order to intelligently assist users in the management of their tasks, we must provide a tool capable of adapting to the nuances of individual users. We handle this by applying techniques found in the area of Machine Learning to the problem. Machine Learning provides us with a set of statistical techniques that can turn data into information for application decision-making. It has been applied to build agent systems for related domains including reminders [6], email management [3], and other personal information.

Because TaskMinder is adaptive, a window of possibility is opened to aid users in breaking bad habits (e.g. procrastination). TaskMinder serves an ideal application for employing the ideas offered by the field of Captology. A new sub-field of HCI, Captology is a research program interested in finding ways to make computer technology persuasive thereby changing people's beliefs and actions. Task management is an intuitive fit for captological thinking and design, since at the most fundamental level, a to-do list is useless if it does not help users get work done. Captology sees computers as tools, as media, and as social actors [7]. In the context of to-do lists and other PIM tools, seeing the computer as a tool is obvious, but seeing our system as a machine learning agent suggests that perhaps thinking of the system as a social actor will be fruitful as well.

DESIGN DETAILS

Our goals with the TaskMinder agent were manifold. We sought to build a system that would actually help users manage their time better by suggesting, either explicitly or implicitly, the right thing at the right time. The high-level goals for the system are as follows:

1. Machine Learning for continuously improving rankings and suggestions of tasks via rich context features.
2. Easy and elegant to-do item entry. Only a single line about the task is required. Other information (due date, importance, urgency) are entirely optional.

3. Conversational interface to provide the ML engine with more data, while being unobtrusive.
4. Multiple operating system capability and future PDA integration.

Context Features

The TaskMinder system is based on the extraction of patterns from the past history of the user and using that data along with the user's current context to suggest tasks that the user will prefer to do in any given situation/time. The system uses four main sources of data to understand the user's "context." They are:

Activity: The system is able to infer the user's level and type of activity based on the number of windows (and therefore programs) that are open on the desktop at a given time. While we did consider using the history of users' activity over a longer period instead of an instantaneous value, the use of this instantaneous value will better support the user in switching tasks, while learning user patterns of activity more effectively.

Location: The system is able to locate users based upon the relative signal strengths they receive from multiple wireless access points of a wireless network (802.11b). For the purpose of our evaluation, we used the Local Area Wireless Network (LAWN) on the Georgia Tech campus. The choice of using this technology was based upon our goal of providing a cheap and efficient location sensing system based on pre existing hardware. Since we surmised that the majority of use of the TaskMinder system would be conducted indoors, GPS would not be a good choice. While this technology will not be able to provide the system with any useful location data while the user is outdoors or in a new place, the system suggestions will gracefully degrade in quality rather than failing completely. Another trade-off made during the design of the system was to merely use the unique IDs of the 3 access points with the strongest signal to triangulate the user's location rather than the use of any clustering algorithm. The reason for this was due to another of our initial goals, namely achieving platform independence. The use of a clustering algorithm would restrict our ability to port the TaskMinder system to handheld and other low power devices. We have however made the architecture of the system flexible enough to allow future additions of any new location sensing technology including GPS.

Time of day: This feature is based upon the time of the day that the user looks at/requests a task. In order to efficiently achieve this we have partitioned the day into 16 discrete bins and every time a suggestion is made the user's feedback is added to the specific bin that represents the

current time. While once again clustering would yield a better partitioning of the system and thereby achieve better results we feel that reducing the complexity of the system in order to achieve platform independence is a better alternative.

User requests: This is a very important source of data for the TaskMinder system as it is directly based upon past suggestions and user feedback. The system keeps a record of not only the type of suggestion that the user requested (i.e. 5 or 15 minute suggestions), but also the user's feedback to the suggestion. This is the main method by which TaskMinder is able to calculate the length of time required to complete a given task. While this feature is very important to the TaskMinder weighting system which is described in a later section, it is not an absolutely necessary component as users may choose to use the system in the list view mode and bypass the request history based weighting system altogether.

User Interface Design

Here we detail the design of the TaskMinder application. The design was intended to be easy to learn and easy to operate. We intended the system to be used on desktop systems. The system has two main functional screens, a "Task Entry" screen, and a "Task List" screen.

The presentation system from the Task Minder is crucial for its functionality. Our main goal is to create an optimal list of tasks and then present it in a way that users can easily and quickly utilize it to *get work done*. A secondary motivation is to create a user interface that lets users provide a host of feedback to the system, feeding our machine learning data repository. A crucial part of this feedback system is that it is transparent to users. TaskMinder does not add specification overload to tasks, but instead uses the natural, conversational interactions, that a user makes with the UI features to collect a rich set of data.

Task Entry: Tasks are entered into the system via a control that is as simple as possible (see Figure 1). The due date control uses screen real estate effectively, trading a bit of increased space for the ease of specifying a date in the current month with only 1 click. Also, we feel that the calendar control provides a better sense of context for the scheduling of to-do items, since it shows weeks and weekends instead of just numerical dates.

There are large radio buttons on the interface that allow users to select an attribute of "urgency and importance" for a given to-do item. The buttons are:

- Urgent and Important
- Urgent
- Important

- Neither Urgent or Important

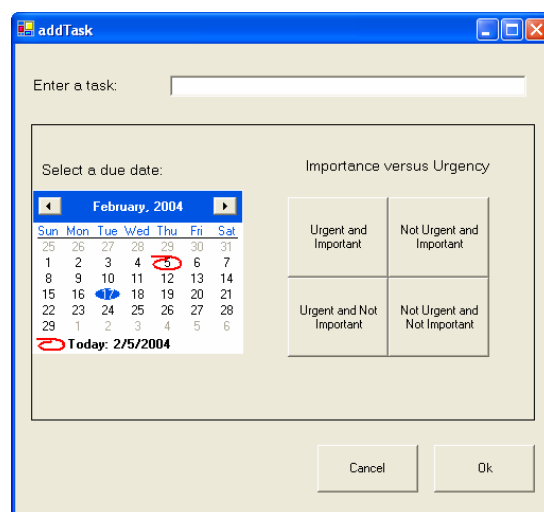


Figure 1: Task entry screen. Tasks are entered via the text box at the top. All of the other fields are optional.

Selecting one of these buttons is optional, but urgency and importance are used in the machine learning algorithm of the system, so it behooves users over the long term to select an element. Further, as a persuasive computing application, we feel that implicitly querying users about the importance and urgency of individual tasks will be beneficial. People often do what is urgent, but not important, and have a difficult time accomplishing important, but unbounded activities. The machine learning algorithm and suggestion system takes this into account and makes suggestions based on more than just urgency (and due date).

To-do List: The task list view is the canonical view of to-do items. It is a list that is ordered in descending order of importance and suggestion from the machine learning back end (to be discussed in the next section). We aimed to double encode importance of tasks based not just on the order of tasks, but also their visual weight. We increase font sizes and bold text based on the weighting algorithm, so that degrees of importance are highlighted non-linearly.

We designed an interface where users can give fine-grained feedback to the system. Because of our constraint on gathering data about when, where, and even why users are performing certain tasks, without needing explicit feedback, or adding time to the interaction, we designed a set of buttons. Our buttons are:

- Done
- Great Idea!
- Today
- Soon
- Later

Our vision with these buttons is that they will be sufficiently conversational elements to appear natural to end users. We did not intend to create an anthropomorphic agent that would use a natural language interface. However, we still strove to make the interface conversational and get past the binary “done” and “not done” distinction apparent in many of the existing task management software packages.

We picked this method for the following reasons. We hope to subtly increase interaction with the system – if users have a way to interact with the to-do items without marking them complete (i.e., to tell the system their plans about when a given to-do item will be done), this will increase the efficiency of the machine learning algorithm. Further, we hoped to give just a hint of personality to the system. The buttons are a reasonable first step toward this, without being saccharine or overdone.

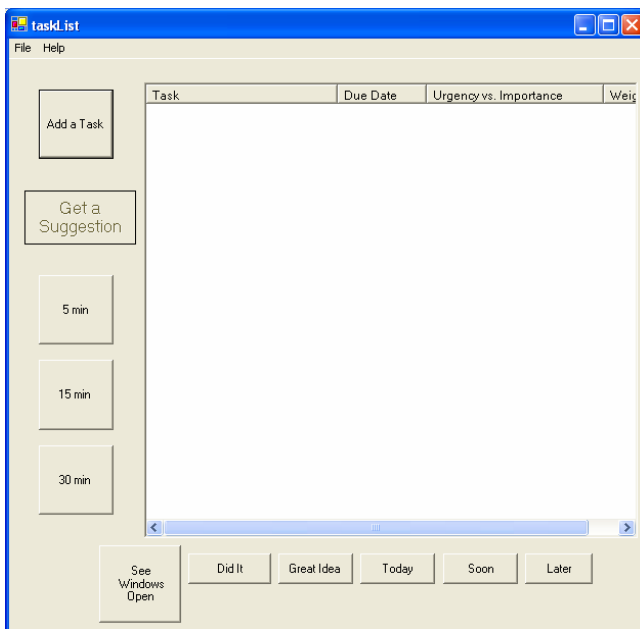


Figure 2: The To-do List. The task list provides a small range of buttons for interaction with the tasks, instead of the binary “done” and “not done.”

Task Suggestions: The TaskMinder system also supports the suggestion of tasks. We believe that users check their to-do lists when they have downtime between meetings or appointments. With this in mind, our interface supports a way to query the system for tasks that might take 5, 15, or 30 minutes (see Figure 2). TaskMind, with its model of historical user behavior and its model of activity and context can predict which tasks might take a certain number of minutes, and bring those to the top of the list.

In conclusion, the interface is fast and elegant in its operation. It supports browsing and visual filtering and has only a few interface elements and interactions for simplicity. The interface prominently displays tasks themselves. The interface and, though the system will learn

about the user and will reorganize, reorder, and refresh the task list, the interface itself will remain constant throughout its operation.

System Design

In the preceding section, we detailed how TaskMinder appears to the end user. We now describe the agent from the back-end. The TaskMinder system is based on a client-server model and uses the Microsoft .Net framework Web Services framework to allow a single server to serve multiple clients of different devices.

Client: The TaskMinder client is a lightweight application whose main functions are to gather details about the user’s context and also to provide the user with a simple and easy to use GUI. The context gathering is done in 2 main ways—the system keeps track of all the currently open programs and also calculates the users location based on the signal strengths of the access points on the wireless LAN. The open programs are found using standard Windows API calls while the access point signal strengths are found using the Site Monitor feature of the Dell TrueMobile 1150 client manager for the Dell TrueMobile wireless system [4].

While the client application currently runs on a laptop computer and as such has a lot of computing power at its disposal we are utilizing only a small fraction of it and offload most of the processing work to the server.

Server: The TaskMinder server runs on a remote desktop machine and can be accessed by the client as a web service. The server not only stores the data collected from the system but it also does the processing required to rank the tasks according to their importance. The system uses a Microsoft Access database and utilizes machine learning algorithms to dynamically weight each task based on the context parameters and user requests it receives from the client machine.

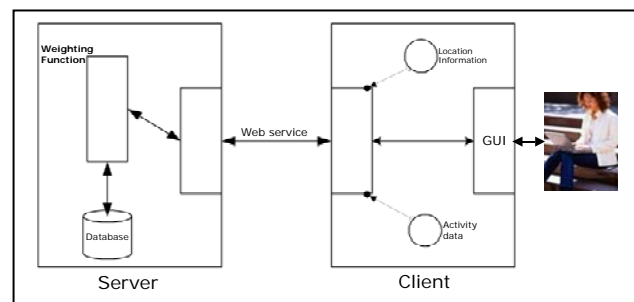


Figure 3: The client-server architecture of the TaskMinder system is shown.

Webservice: The client-server model was specifically chosen for this task so that a single user may use multiple clients without having to duplicate his tasks onto each one. The implementation is based on open standards such as

XML and SOAP so that clients may be written for different platforms with the sole necessity that they be able to interpret these open standards.

These include personal computer based clients (Windows, Linux) as well as mobile platforms such as J2ME and WinCE. All devices need the basic HTTP capability and a toolkit for parsing XML and/or interpreting SOAP requests and responses.

The specific implementation platform chosen was the Microsoft .Net framework for both, the client and the server. The server is a .Net Web Service, whereas the client is a VB.Net desktop application serving as a Web Service client. Interaction between the two has been designed to minimize round trips (thus, multiple items of data are sent to the client in one web service call.) We do assume limited processing power on the client (e.g. the function of sorting tasks based on weights is delegated to the client, for two reasons: reducing round trips, and better interactive response.)

Database: The database of the TaskMinder server stores all the tasks that the user has ever entered and also the weights associated with them. The structure of the database is shown in figure 2. The database basically stores data in terms of words; each task is decomposed into its constituent words, each of which has its own database entry. Associated with each word is a count of the number of times the user has given the system a task that contains the word and a bias vector which shows the relative importance of the word. The words also have associated with them a set of specific circumstances which allow TaskMinder to go through the past history and suggest the task the user is most likely to complete. Each circumstance also has a count of the number of times the user has given it feedback and a weight which is recalculated every time the user gives it feedback.

Weighting Algorithm: The weighting algorithm of the TaskMinder system uses a count of the number of times each circumstance has occurred along with a word bias vector to calculate the weight of a word in a given state. A random variable whose value is dependant on the count of the state and also the word is also added to the weight in order to achieve a balance between exploration and exploitation.

$$\text{Word}_x = \alpha_x W_{xb} + \alpha_{x1} W_{x1} + \alpha_{x2} W_{x2} + \dots + \alpha_{xn} W_{xn}$$

α is a constant which is based on the count

W_{xb} = Bias vector of word x

W_{xn} = Weight of word x in circumstance

After the weight of individual words is calculated the task weight is calculated by averaging the weights of the individual words (see Figure 4)

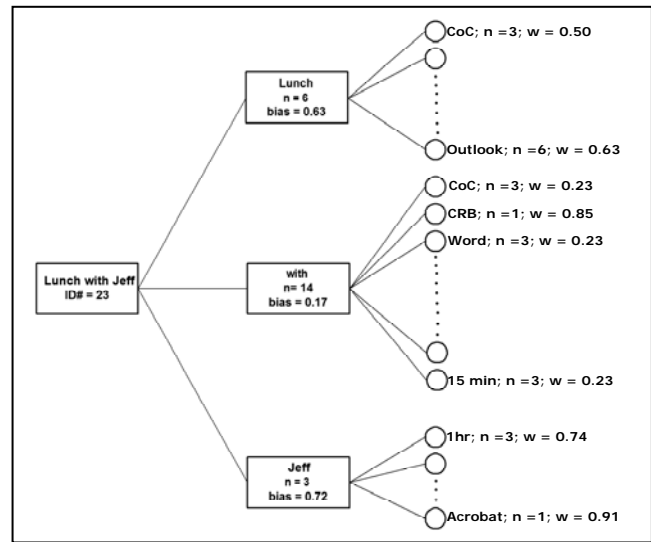


Figure 4: The weighting Algorithm as it parses a to-do item. One can see the weights on each of the constituent words and example feature data.

System Operation

We walk the user through a scenario, explaining the interaction between the system components described above. Say the user asks for a five minute suggestion. The TaskMinder system's primary mode of operation is when the user asks the client application for a suggestion of what tasks to do in next few minutes.

The users must include the amount of time they have free so that the system can tailor the suggestions to fit in that time frame. Once the user makes a request for say a 10 minute task the client immediately collects information about the current state or "context" of the user based on location, time, and activity level. It then passes this information along with the request time frame to the server which calculates the suggestions to be made.

The server on receipt of the information first goes through the task list to extract the tasks that are yet to be completed. Next the system decomposes each task into individual words that can be found in the database and have bias weights associated with them. The server's next task is to search through the list of "circumstances" associated with each word to see if there is any match to the present state - an exact match is not required and a match to any of the current variables is sufficient. Once the matches are found the weights associated with the match are added to the bias vector and if there is no match a default value is added. A random value is added as well whose maximum limit is determined by the number of times both the word and the state have occurred before - this is needed to balance between exploration and exploitation.

The weight of each task is calculated by averaging the weights of the words that compose the task and task list is then ordered according to the calculated weights. The server then transmits the ordered list back to the client which displays the list.

Once the user has given his/her feedback about a suggestion (i.e. Done, Great Idea!, Today, Soon, Later) the client once again sends the server the current state along with the user's response. This is done to allow the server to update the weights of the both the circumstance and the word bias to take into account the users feedback. A click on the done button removes the item from the list, the others just re-weight the particular task, which may or may not cause a reordering of the to-do list.

EVALUATION PLAN

We have just begun to evaluate the TaskMinder system. The plan is to have a small number of users over a long period of time use the system and provide feedback on its features. We are especially interested in the machine learning aspects, how well the system learns user patterns and makes "intelligent" suggestions and rankings (from the participants's perspective) . As well, we are curious about the interface decisions that we made and how they might impact use of the system. We've designed a daily questionnaire that participants will fill out that has likert-scale questions as well as more open-ended questions. We care very about which kinds of tasks (important ones, urgent ones) get into the system and the only way to determine this is in a longitudinal evaluation. The system is also being "bugged" in some ways so that the authors can get reports about how much interaction users have with the system, and the character of that interaction. We plan to compare these two to see how much user-reports match with system data.

FUTURE WORK

The TaskMinder system is currently deployed on laptop computers. The deployment decision was made on expediency factors such as availability of hardware, the constraints of other research software that is being used in the application (the location sub-system only works with certain models of wireless network cards) and constraints of time. Our first order of work is to port the system to PDAs. The system was designed to be multiple client aware, and the client-server architecture was designed with portable devices in mind.

The PDA implementation of the TaskMinder application will have the same functionality as the current system. It will connect to the server and get location context over the wireless network. We will redesign the user interface components for mobile applications, taking in to account the reduced screen size and conventions of the platform [5]. We will be featuring the suggestion functionality more

prominently, as well as making interacting with the task list more compact as well as faster.

Other areas where we'd like to bolster the system include further explorations of context, and changes to the machine learning algorithm. The contextual information that TaskMinder gathers provides a reasonable model but we'd like to ascertain which current features are the most predictive for user modeling (and whether this predictive power is consistent across users). This information is something that can be determined over a longer evaluation. We also hope to be in a position to try out a slightly modified machine learning algorithm in the course of a longer-term evaluation. We will be modifying the system to store not just single word vectors, but double and even triple word vectors, to see if they have an impact on learning or on suggestion performance.

CONCLUSIONS

We have designed and implemented TaskMinder, a much improved to-do list agent. The agent uses user location, activity and history to produce a prioritized list of tasks and to suggest tasks. The system learns via a statistical machine learning that has environmental context features as well as user-history features. The user-history is gathered by a conversational interaction model provided by an intuitive and efficient user interface.

ACKNOWLEDGMENTS

We wish to acknowledge Charles Isbell, Jeff Pierce, and the whole of the CHI community. Also, our moms.

REFERENCES

1. Aberdeen Group *"Mobile Handheld Devices: Enabling Enterprise Communications and Data Management,"* (2001)
2. Avrahami, D., Hudson, S., Moran, T. P., and Williams, B., " *Guided Gesture Support in the Paper PDA* ", Proceedings of the 2001 ACM Symposium on User Interface Software and Technology (UIST 2001).
3. Boone, G. *Concept features in re:agent, an intelligent email agent*. In Second International Conference on Autonomous Agents, may 1998.
4. Castro, P., Chiu, P., Kremenek, T., and Muntz, R., *A Probabilistic Location Service for Wireless Network Environments*. Ubiquitous Computing 2001, Atlanta, GA, September 2001.
5. Decker, K. *Coordinating Human and Computer Agents* . In W. Conen, and G. Neumann, editors, *Coordination technology for Collaborative Applications - Organizations, Processes, and Agents*. LNCS #1364, pages 77--- 98, 1998. Springer-Verlag.

6. Dey, A., and Abowd, G., *CybreMinder: A context-aware system for supporting reminders*. In Proceedings of Second International Symposium on Handheld and Ubiquitous Computing, HUC 2000, pages 172-186, Bristol, UK, September 2000. Springer Verlag.
7. Fogg, B., *Persuasive Computers: Perspectives and Research Directions*. Proceedings of CHI 1998.
8. Horvitz, E. (1999). Principles of mixed-initiative user interfaces . CHI 99, 159-166.
9. Marmasse, M., and Schmandt, C. *Location-aware information delivery with comMotion*. In Proceedings of International Symposium on Handheld and Ubiquitous Computing, 1999.
10. Peters, E., *Organize This! Investigating Personal Information Management Practices*. 2001, Georgia Tech, Atlanta, GA.
11. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. *The context toolkit: Aiding the development of context-enabled applications* . In CHI, pages 434-441, 1999.
12. Taskline. <http://www.taskline.info/screen.asp>.
13. Wong, B., Starner, T., McGuire, R., *Towards Conversational Speech Recognition for a Wearable Computer Based Appointment Scheduling Agent*. Tech Report. College of Computing Atlanta, GA, 2002.