TaskPort: A Task Management Interface in an Intelligent Cognitive Assistant System

Yili Wang
School of Computer Science
Carnegie Mellon University
yiliw@andrew.cmu.edu

Advisor: Prof. David Garlan
School of Computer Science
Carnegie Mellon University
garlan@cs.cmu.edu

Abstract

People are spending an increasing amount of time handling everyday routine tasks in their daily lives. To help people work more efficiently, Carnegie Mellon University is currently developing the RADAR (Reflective Agent with Distributed Adaptive Reasoning) system, a software-based cognitive personal assistant. The task manager is an essential component of the RADAR system that manages high-level tasks and coordinates communications among other intelligent assistants in a personal space. The task manager interface is an application that would not only allow the users to browse, create and modify tasks, but also provide more intelligent functionalities regarding task management. In this paper, we discuss the motivation, and an implementation of such task manager interface, called TaskPort. To better understand the elements that provide an edge in efficiency and productivity for the user, we explore various techniques used in the task manager interface and attempt to resolve several interesting challenges, such as flexibility, usability, scalability, and integration.

1. Introduction

As more and more communications among people are done electronically, computer users are pressed to keep track of a growing number of everyday routine tasks, such as answering emails, scheduling meetings, allocating space and resources, or updating various personal or project websites. Alone, such tasks are easy to accomplish and do not require too much time. However, the process of managing a large amount of these tasks becomes repetitive, confusing, and even chaotic at times. Moreover, these tasks may also vary greatly in complexity and time span. For instance, coordinating a large project meeting takes weeks or even months to complete and requires handling myriad of information sources and dozens of people, whereas answering a personal email takes only a few seconds of the user's attention and needs no additional pieces of information at all. Consequently, task management need to account for such change in complexity and thus making the problem even more difficult and less efficient.

The issue of task management is especially relevant to people whose jobs consist mainly of interacting with other people, for instance, managers, professors, and salespeople. Imagine handling tens and even hundreds of emails everyday with varying degrees of importance and settings, negotiating many meeting places and times with colleagues and clients, heading or participating in various projects that have very different requirements and due dates. It is crucial for those who work in a collaborative environment to be able to not only monitor status of the currently active tasks but also find tasks from the past, such as "the last time I a meeting with Mr. X", or "the presentation I gave two months ago". This is more than merely remembering where that presentation is stored on disk or when that meeting took place. This is a matter of finding the entire process of how the task was completed – the steps taken to accomplish the task and any artifacts resulted from the task. The problem is worsened since these people tend to work in multiple environments as well, moving from clients to clients, or offices to homes. As they move from location to location, they must be able to carry their tasks with them as well. Efficiency is greatly hindered when they do not have access to their tasks or the ability to track their progress.

People currently deal with task management in many different ways. Some like to leave a variety of post-it notes, each filled with important task information. Others like to write extensive to-do lists either on paper or in notepad that may be organized according to their priority, category, and the people involved. Some even organize pieces of task information in various folders according to some form of categorization. These methods often serve as good reminders for people and what they need to accomplish. However, they certainly are not sufficient as task management tools. Any information written on loose papers could be lost. Even when we assume all task information is well kept, these methods do not offer capabilities to access all task related information and to check current status beyond the most basic levels.

There are a few commercial products on the market today that are designed to assist users in managing their tasks. These tools are software-based task list or to-do list that are often offered together with other calendar or email applications. Some examples are To-Do List functionality in Microsoft Outlook and Oracle Collaboration Suite. However, these tools are mostly one-dimensional offering just yet another way for users to record reminders and do not have much flexibility in their functionalities.

One better solution to the problem is to hire as many personal assistants or secretaries as one needs in order to manage all his / her tasks. These assistants must be not only accessible at all times but also knowledgeable about all task related information, such as one's calendars, contacts, emails, other relevant information, as well as one's personal preferences. Evidently, this solution is not plausible in all situations or even at all in some cases. Thus, an interesting research problem arises: how do we effectively manage our overflowing tasks without the help of these all-knowing ubiquitous personal assistants?

The main goal of our research is to provide a framework for an interface that allows users to efficiently manage all their tasks, specifically a system that offers scalability, flexibility, and usability. Our task management interface has been developed as part of the RADAR project at Carnegie Mellon University. RADAR takes advantage of the fact that many of the user's routine tasks are highly automatizable and uses advance technologies in machine learning, knowledge representation and cognitive systems to create a software-based cognitive personal assistant that provides a solution to the problem posed above. In this paper, we first introduce the RADAR system, the functionalities of the various components including the role of the task management interface. We then explore the specifics of the problem domain in designing a task management interface. We finally proceed to explain the solution offered by our design and its usage and implementation.

2. Related work

<Empirical investigations, agent research, etc.>

3. What is RADAR?

Reflective Agent with Distributed Adaptive Reasoning (RADAR) is a large five year project developed by Carnegie Mellon University. It aims to create a software-based personal assistant, making extensive use of cognitive systems technology. It not only fully understands the user's tasks, preferences, and the changing environment, but also anticipates and fulfills the user's information needs. It greatly increases the user's work efficiency by automatically handling some of the user's routine tasks and provides suggestions for the user on others. The system also intelligently adapts to the user's behavior over time and is capable of handling unexpected situations and requests without reprogramming. Consequently, such cognitive personal assistant is capable of filtering user emails, scheduling meetings, and allocating resources in a way that the user would do so.

RADAR is built upon the idea of "intelligent agents", or **tasklets**, each specializing in a particular area. CMRadar project, for instance, is an "intelligent assistant" software that specifically tackles the problem of calendar scheduling where traditional tools such as Outlook and MeetingMaker are passive tools that require significant user attention. It aims to produce an intelligent and more active calendar scheduling system that possesses the capabilities to automatically schedule and negotiate preferred meetings, read and respond to routine email meeting requests, and in general greatly reduce the time overhead of meeting scheduling. There could be other tasklets as well. For example, a Webmaster Tasklet would actively assist users in updating a particular project website on a monthly basis; an Email Tasklet would intelligently organize one's emails according to the people and events involved and filter out unwanted spam. The grand challenge of RADAR is to be able to enable organization after an unexpected crisis. In a crisis situation, the system will have to handle a flood of messages that require attention and response, to schedule urgent meetings, to solve problems outside the usual boundaries, and to perform even better next time.

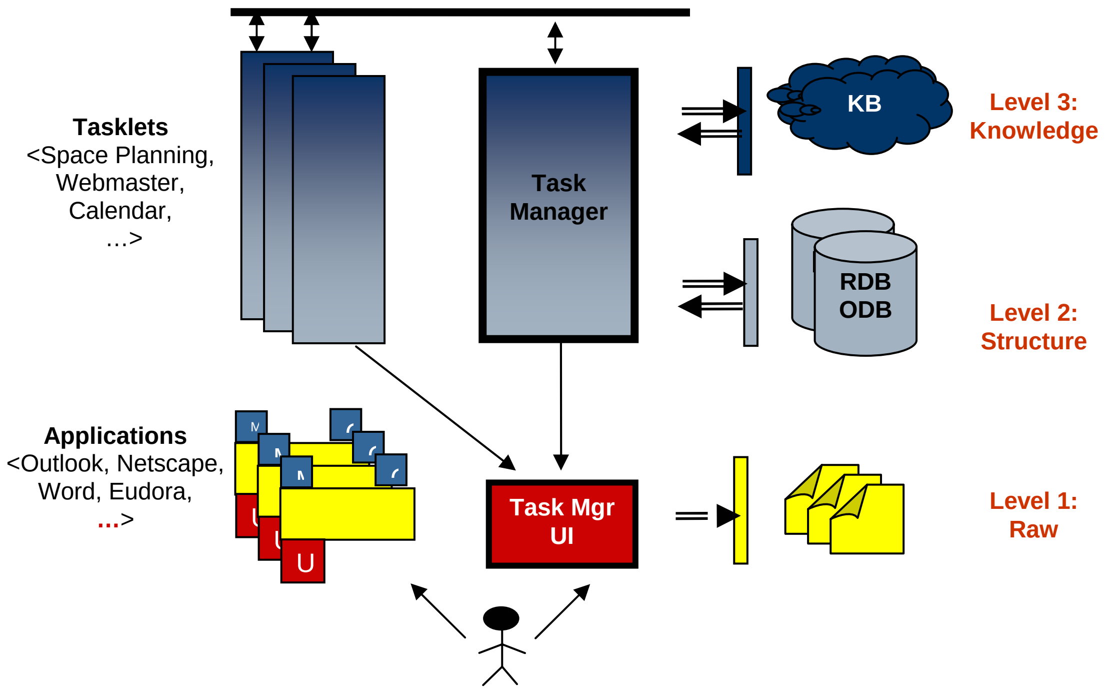


Figure 1 Basic architecture of RADAR system.

Figure 1 is an illustration of various components in the RADAR system. Information is categorized into three separate levels: raw, structured, and knowledge based. Most commercial products today exist only at the first level. Software applications such as Microsoft Outlook, Word, and Eudora deal directly with file systems and stores emails and other information as raw files. RADAR system takes task management to the next levels by enabling storage for user plans, goals, preferences and strategies in additional to the traditional storage. This additional RADAR information can be further categorized as either structured or knowledge based. Intelligent agents (tasklets) such as CMRadar access and enrich such information while actively assisting users with specific types of tasks.

Even though tasklets, working separately, can be considered specialized personal assistants, they cannot solve more complex problems. Most tasks in the real world require cooperation and communication between one or more tasklet. A simple example of a complex task is meeting scheduling. In order to schedule a meeting between several people, the organizer needs to first find a time that is available to all participants. Then, the organizer also needs to find a room that is available at the meeting time that is suitable for the meeting. You can easily imagine that a calendar tasklet may be able to accomplish the first part of the task by negotiating an appropriate meeting time for all meeting participants. However, in order to find an available room at the meeting time, we need to use a space planning tasklet. Now the question becomes how these individual tasklets would communicate with each other to accomplish a common task.

The solution to the problem in the RADAR system is the component called Task Manager (shown in the middle of the Figure 1). The task manager provides communication, coordination, and basic notification functionalities among tasklets for a single or multiple users. It keeps a relational structure database of all tasks for the user. The database not only includes high level information regarding the tasks, such as task description, due date, and importance level, but also maintains records of tasklets that are responsible for these tasks. The task manager may also maintain a knowledge base library that overtime learns from the user's behavior and intelligently adapts to the user's preferences. Furthermore, the task manager is the central point where multiple users can interact with each other. In the RADAR system, every user is assumed to have a task manager, but not necessarily all the available tasklets. Therefore, task manager takes on the responsibility of coordinating efforts in accomplishing various tasks between multiple users where the available tasklets for each user may differ.

So far, we have seen how the RADAR system can intelligently automate and assist certain routine tasks, regardless of their complexities. One important piece missing is how the users would easily interact with the system in a manner that is most efficient for them. Individual tasklets may provide user interfaces by extending existing functionalities of applications such as Microsoft Outlook. It is also crucial for the task manager to have its own user interface that would help the users manage all their tasks and communicate with the tasklets. The task manager interface component (shown in the center bottom of Figure 1) would provide these essential functionalities that allow users to easily access their various tasks and manage the tasklets through the task manager backend. In the rest of the paper, we will focus on the development of task manager interface and the various challenges encountered in its development.

4. Task manager interface problem domain

In this section, we will describe the basic concepts and requirements for developing a task manager interface.

From a user's perspective, we feel that there are five basic areas that a task manager interface must address. First, the task manager interface must have the **flexibility** to allow users to maintain tasks regardless of their level of complexity, time span, and current status. As we mentioned before, traditional commercial task lists fail because they are one-dimensional applications that are only suitable for maintaining a small amount of generic tasks. These applications do not distinguish between tasks with varying levels of complexities or time span. They are merely a poor alternative to keeping hand written reminders.

Second, the task manager interface must be **user friendly** and be fully aware of what the user is trying to accomplish. It should be helpful and provide functionalities that would assist the users in monitoring and accomplishing their tasks. People continue to choose hand written to-do lists and other traditional methods for task management over commercial tools because these tools are far less convenient for the users and present little additional support or value for their users. Any implementation of the task manager should address the problem by offering convenience and useful functionalities to the users. This concept of usability also extends to the notion of accessibility. For users who often work in multiple environments, they should be able to access their tasks and obtain the most current status from all locations.

Third, the task manager must address the problem of **scalability**. The task manager needs to be able to effectively handle thirty tasks as well as thirty thousand. The issue of scalability not only applies to performance in a traditional sense but also relates to representation of the tasks. A naïve way of simply listing tasks in a table may be scalable in terms of performance. However, this simple design suffers from the lack of ease for navigation and clarity when representing a large amount of tasks to the user. On a day to day basis, most users only care about their most relevant and active tasks, such as homework that they need to complete at the end of the week or meetings that they should schedule for the project that they are actively participating. Yet, they would also need to reference events and tasks from the past in order to remember the exact steps taken to accomplish a certain type of tasks. The task manager interface should take into account how people would likely to use the application when addressing the problem of scalability and provide a suitable solution that allows not only actively monitoring of only the most relevant tasks but also the ability to view all previous tasks if necessary.

Fourth, the task manager interface must be easily **extensible** as the architecture of the overall RADAR system changes and new functionalities are developed. The structure of the interface must be flexible so that existing components can be improved and additional components can be added without too much change in the existing code.

Finally, because the task manager interface is developed as part of the research effort of the RADAR project, it must provide seamless **integration** to the rest of the RADAR system as illustrated in Figure 1. The task manager interface must actively communicate with the task manager backend to not only obtain the most current information regarding user tasks and tasklet availability but also relay user changes in the tasks to the task manager so that the information could be passed to other interested parties. It should also provide capabilities of communicating with tasklets in order to attain additional information on automated tasks that is not directly stored in the task manager. Moreover, the task manager interface should adhere to the basic principles of the RADAR project. That is it should employ technologies to enable the development of a software-based cognitive personal assistant that is capable of improving itself from adapting to the user's preference or behavior.

From a functionality standpoint, the task manager interface should provide both basic and more advanced tools for the user. At the very least, the interface should allow the user to browse and modify all their tasks and be able to create new ones. It should be able to send and receive messages regarding to changes in these tasks (updates, creation, or removal). It should also provide appropriate response in case of error or failure with the system. Additional feature can be added to increase user productivity and usability.

5. An implementation for task manager interface

In this section, we describe a prototype system for task manager interface that we have developed over the past year. This implementation attempts to address the problems described in the previous section or to provide the first steps towards that direction. Technical details regarding the implementation are given in Section 7. We will first discuss the basic functionalities of the prototype system and then describe how the system deals with the issues of flexibility, usability, scalability, and integration. Keep in mind that this prototype for task manager interface has been developed partly independently from the task manager backend and rest of RADAR. Since we are still in the early development stage for the RADAR system, many components are still merely black boxes. Some features in this prototype are still not integrated with or even supported by the larger system. However, these features are shown and described in this paper as how we envisioned the user interaction would be like with the task manger interface after these capabilities are completed in the backend. The main goal of implementing these features is to explore those characteristics of the interface that would increase user productivity and work efficiency as well as to present a proof of concept. I will denote these unsupported or unintegrated features with the symbol "*" in the following sections.

5.1. Defining user tasks

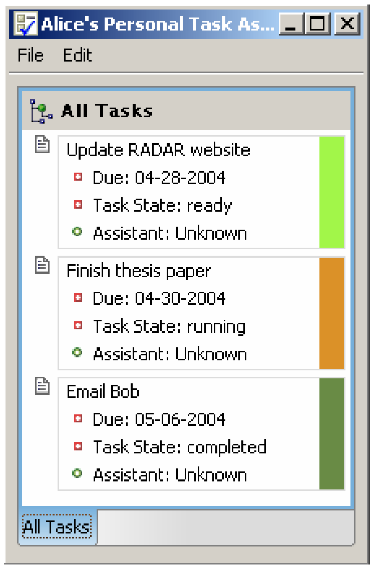
A **task** in the task manager is defined to be any everyday activity of the user such as scheduling a meeting, preparing a presentation, updating a project website, or writing a paper. A task may involve one or more participant and may contain of one or more subtasks as well. Table 1 shows the various task attributes stored in the task manager. The listed task information is stored in a relational database on the task manager backend and can be accessed by the task manager interface.

Table 1. List of task attributes

Id	Unique identification number of the task
name	Name of the task
description	More detailed description of the task
state	Current state of the task (new, ready, running, completed, suspended, failed, canceled or deferred)
importance	Importance of the task (highest, high, normal, low, or lowest)
startDate	Starting date of the task
dueDate	Due date of the task
createDate	Date when the task is originally created
endDate	Date when the task is officially ended, either completed, canceled or failed
lastUpdateDate	Date when the task is last updated by the user or any responsible tasklet
ownerId	Unique identification number that indicates the owner of the current task
parentId	Unique identification number for the parent task in cases where the parent task contains subtasks

5.2. Understanding basic functionalities through a simple scenario

The basic functionalities of TaskPort include 1) browsing, 2) creation, and 3) modification of a user's tasks. Users must be able to browse or modify all tasks whether it is current or from the past and also be able to create new tasks. To familiarize with the basic flow of the application we present a simple scenario that involves a day in the life of Alice, who is a student at Carnegie Mellon University.



Update RADAR website
- Due: 04-28-2004
- Task State: ready
- Assistant: Unknown

Finish thesis paper
- Due: 04-30-2004
- Task State: running
- Assistant: Unknown

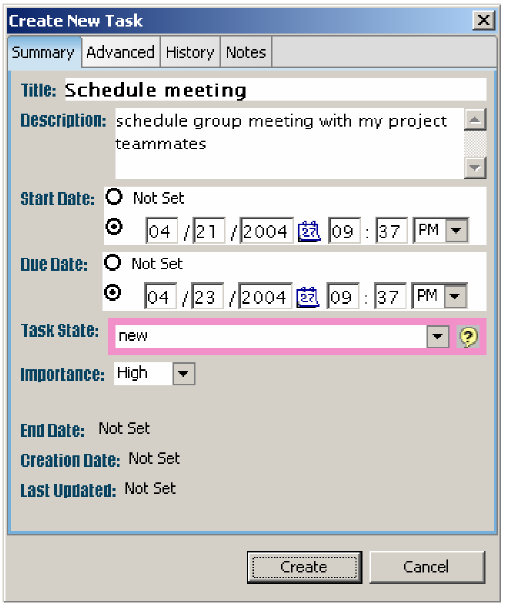
Email Bob
- Due: 05-06-2004
- Task State: completed
- Assistant: Unknown

Figure 2. Screenshot of Alice's personal task assistant

Alice starts the day by logging into TaskPort and examines her list of tasks. Figure 2 is a screenshot of the application that Alice's sees. Alice currently has three separate tasks: 1) update RADAR website, 2) finish thesis paper, and 3) email Bob. Under each task, various task attributes are also displayed. These include the task due date, the task state, and the assistant that is currently responsible for the particular task. The colored bars on the right side represent a quick way of identifying the state of a task. In this case, Alice chooses to use light green to represent "ready" state, orange to be "running" and dark green to be completed.

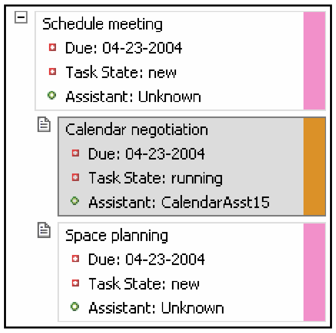
After viewing all the current tasks, Alice decides to create a new task for scheduling a group meeting with her project teammates. Figure 3 displays a short form that she fills out in order to define the new task. A similar form is used for modifying a task. The tabs "advanced", "history", and "notes" are more relevant for task modification than for task creation. When Alice clicks the "create" button, a createTask message is sent to the task manager backend and a new task with a new unique task id is created accordingly on the task manager database. Ideally, the RADAR system would be able to parse the title and description of the task definition and would in turn search a knowledge-based library to decide whether this particular task could be broken down into a series of subtasks. For instance, "schedule group meeting" is a complex task that could be broken down into "negotiating a meeting time" and "finding available room for the meeting". Assume that this intelligent component does exist, then TaskPort receives the subtasks information from the task manager backend and displays the newly created task and subtasks in its browser (shown in Figure 4). For now, since intelligent black box is not implemented, Alice could either manually create the subtasks or leave the new task as is.

After creating the new task of scheduling a group meeting and its subtasks (either automatically or manually), Alice is ready to start one of the subtasks, which is to negotiate an appropriate meeting time with her teammates. She accordingly changes the state of that subtask from "new" to "ready" (shown in Figure. 4). Negotiating meeting times among multiple people is the specialty of calendar tasklets. All calendar tasklets register their interests with the task manager backend so that they would be notified when a new calendar type task is created. Alice checks her Tasklet Manager to see which tasklets are currently available. She sees that besides her personal task assistant she also has a calendar assistant that is currently operating. Thus the calendar tasklet receives a message from the task manager backend about the group meeting subtask and decides to become the responsible tasklet. Alice's calendar tasklet changes the task state for meeting time negotiation again from "ready" to "running". Consequently, TaskPort receives the status change message through the backend and updates its display in the browser. The assistant attribute is changed as well as the task state attribute. We can see from Figure 4 that "CalendarAsst15", the tasklet id for Alice's calendar tasklet, is now displayed as the assistant.



Title: Schedule meeting
Description: schedule group meeting with my project teammates
Start Date: Not Set / 04 / 21 / 2004 09 : 37 PM
Due Date: Not Set / 04 / 23 / 2004 09 : 37 PM
Task State: new
Importance: High
End Date: Not Set
Creation Date: Not Set
Last Updated: Not Set

Figure 3. Screenshot of dialog for creating a new task



Schedule meeting
- Due: 04-23-2004
- Task State: new
- Assistant: Unknown

Calendar negotiation
- Due: 04-23-2004
- Task State: running
- Assistant: CalendarAsst15

Space planning
- Due: 04-23-2004
- Task State: new
- Assistant: Unknown

Figure 4. Created new tasks and its subtasks in hierarchical display

Ten minutes later, Alice decides to check on the status for the meeting scheduling task. However, she notices that the task state for "calendar negotiation" subtask has been changed to "suspended". Feeling slightly concerned and confused about the situation, she double clicks on the task attempting to find some explanation for the task suspension. Under the "advanced" tab, Alice sends a request message to the task manager in order to retrieve some status information from the calendar tasklet. After attaining that information, task manager sends a respond message back to TaskPort*, stating that the task state has been updated to "suspended" because a teammate Bob's calendar tasklet cannot be contacted and Alice's calendar tasklet is currently waiting for a response (see

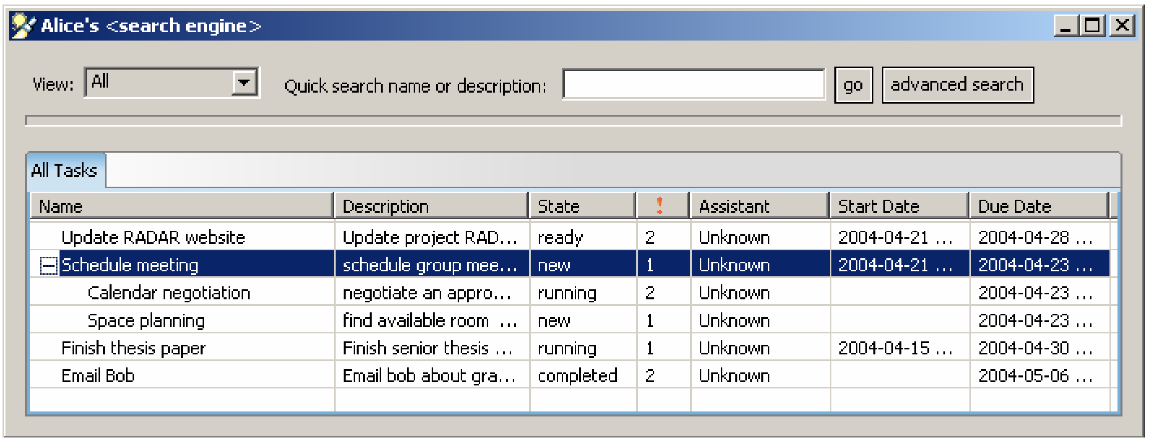
Figure 5). Relieved after seeing that status report, Alice returns to her other work.

5.3. Establishing flexibility

As described in the previous section, traditional task management tools are not flexible in their definition of a task as well as how they display tasks. TaskPort overcomes this problem from both protocol and implementation levels. At the protocol level, our definition of a task is different from the traditional sense because it allows task hierarchy and task states. A hierarchy task structure is imperative to supporting tasks with varying complexities. We have already seen a good example of this in the simple scenario presented above. A very simple task such as “email Bob” would not have any subtasks whereas a more complex task such as “schedule meeting” has two subtasks. Task state, in addition to a hierarchical structure, enables TaskPort to represent tasks that are not merely reminders and span over a longer period of time and involve multiple steps and stages. When the change history of task state is recorded, it creates a valuable journal of how the task is eventually accomplished or failed. The user may look back any time in the future and receive an exact account about the task. This may be useful for the user to successfully accomplish a similar task in the future or to avoid a mistake made in the previous task.

At the implementation level, TaskPort presents several unique methods of displaying user tasks and thus extending the idea of flexibility. TaskPort incorporates the concept of a **taskbin**, which is simply a collection of tasks that share some similarity. For instance, we can create a taskbin to contain all tasks that are due today. Since people are already familiar to the idea of categorization, the idea of taskbins should be intuitive. Taskbins allow users to organize tasks and view them in different categories, just as how they save files in various folders. Taskbins may also maintain a hierarchical structure where subgroups of tasks could be created within a taskbin. Moreover, taskbins may overlap. In other word, any particular task is allowed belong to one or more taskbins. The underlining reason for allowing overlapping taskbins is that intuitively, a task may share similarities with different sets of tasks when different attributes are considered. The user, for example, may want to see both the tasks that are due today and currently suspended tasks. If a task is suspended and is also due today, then it will appear in both taskbins.

Furthermore, TaskPort supports two perspectives for viewing tasks, dashboard view and search engine view. Figure 1 illustrates the dashboard view of the task. The dashboard is designed to be condensed and compact and only occupies a small portion of the user’s desktop. One disadvantage of using a full blown application such as Outlook is that it requires the entire desktop; it is difficult to monitor the status of tasks while doing other things on the computer. The dashboard view is inspired by the look of desktop post-it notes, which are small workspaces that people use to keep reminders and other notes. The dashboard inherits the compactness of post-it notes and yet provides a more organized view of tasks rather than just reminders. It should be used to display only the few most relevant tasks.



Name	Description	State	!	Assistant	Start Date	Due Date
Update RADAR website	Update project RAD…	ready	2	Unknown	2004-04-21 …	2004-04-28 …
Schedule meeting	schedule group mee…	new	1	Unknown	2004-04-21 …	2004-04-23 …
Calendar negotiation	negotiate an appro…	running	2	Unknown		2004-04-23 …
Space planning	find available room …	new	1	Unknown		2004-04-23 …
Finish thesis paper	Finish senior thesis …	running	1	Unknown	2004-04-15 …	2004-04-30 …
Email Bob	Email bob about gra…	completed	2	Unknown		2004-05-06 …

Figure 5 Search engine view of the same tasks as shown in Figures 2 and 4

search engine, on the other hand, employs a more traditional look. Tasks are displayed in a table form with support for a hierarchical structure. Figure 5 presents a screenshot of Alice’s search engine displaying the same tasks as in the dashboard view

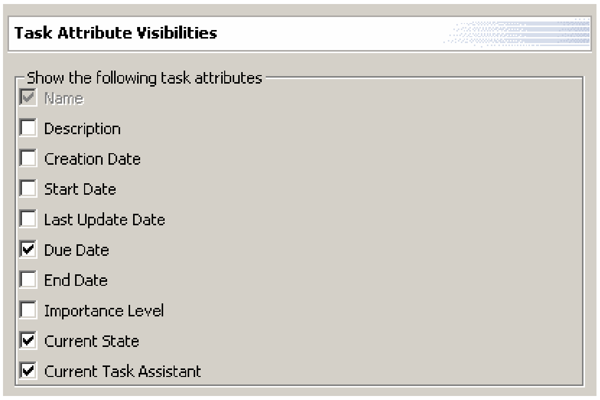
(Figure 1). We can immediately notice that the search engine view is much larger in appearance. However, it facilitates the process of searching, sorting, and filtering on a large set of tasks.

With these two perspectives on viewing tasks, the users may choose either perspective at appropriate times. One way of using these two perspectives together is to keep the dashboard open at all times in order to monitor the most active and current tasks and to utilize the search engine whenever we need to find a particular task in the past.

5.4. Creating usability

Usability applies to the overall utility and convenience that the task manager interface provides to the user. TaskPort aims to understand what the users are trying to accomplish and offers useful functionalities that increases productivity of the user.

User preference is one area that TaskPort focuses on. Evidently, different people prefer different looks and have different ideas about how tasks should be displayed in the user interface. Therefore, it is crucial the any task manager interface to allow users the ability to define the looks of their interface. TaskPort acknowledges this issue and takes the first steps in creating a completely user-defined interface. It currently presents three main areas capable of user customization. First, the user has complete control over how many task attributes are visible in both the dashboard view and the search engine view (shown in Figure 6). In the dashboard view, for example, displaying few task attributes would make the application more compact. On the other hand, displaying more task attributes would show more information in a glance and require the user to click less for more information. Second, the users can define the color codes that indicate task state on the right of the task display in the dashboard view (see Figure 1 to see what the color bar looks like). Alice, for example, defined pink to be "new" state, light green to be "ready", orange to be "running" and so on. Bob, on the other hand, may have a completely different set of colors that are intuitive to him in representing these task states. Third, users can customize which taskbin collections should be shown at startup of the application in the dashboard view. Since the dashboard is designed to display only the more relevant tasks, it is imperative that TaskPort provides a way for the users to define what "relevant" actually means.

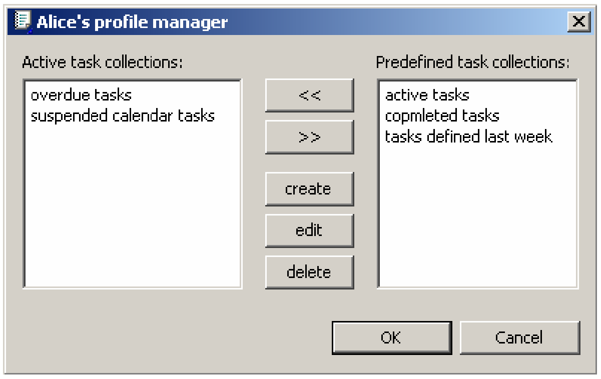


Task Attribute Visibilities

Show the following task attributes

- ✓ Name
- Description
- Creation Date
- Start Date
- Last Update Date
- ✓ Due Date
- End Date
- Importance Level
- ✓ Current State
- ✓ Current Task Assistant

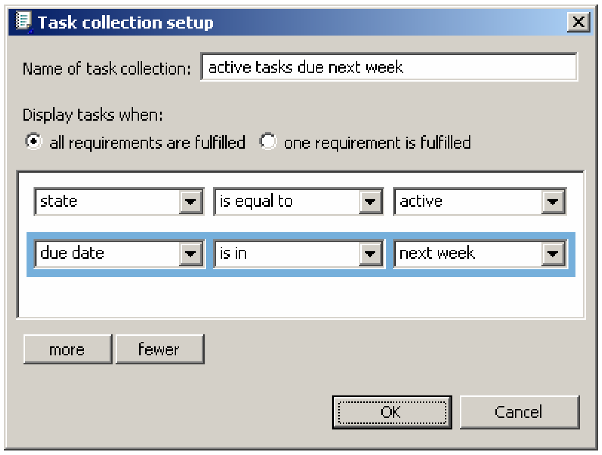
Figure 6. Users have full control over what task attributes are displayed in both the dashboard view and the search engine view



Alice's profile manager

Active task collections:	Predefined task collections:
overdue tasks	active tasks
suspended calendar tasks	copmleted tasks
	tasks defined last week

Figure 7. Screenshot of Alice's profile manager



Task collection setup

Name of task collection: active tasks due next week

Display tasks when:
[✓ all requirements are fulfilled] one requirement is fulfilled

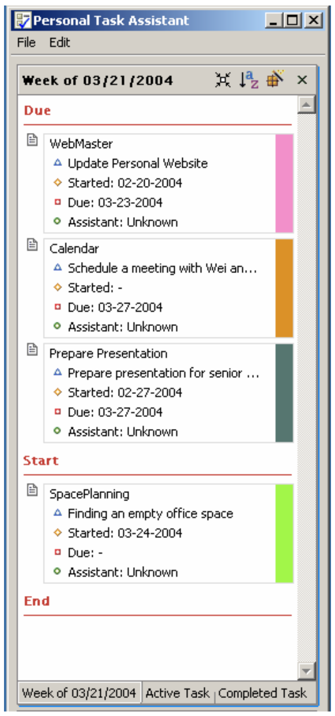
state	is equal to	active
due date	is in	next week

Figure 8. Form used to set up a new task collection

Figure 7 and 8 illustrate how the user could create such unique profile regarding task collections. Figure 7 shows a list of taskbin collections that the user has already defined. The user may create a new collection, edit existing ones, or delete any defined collections. The user can also choose to include any of the predefined task collections shown on the right side. In this case, Alice has already defined two taskbins for her dashboard: 1) tasks that are overdue and 2) calendar tasks that are currently suspended. Figure 8 shows the setup menu for creating a new collection to display all active tasks that are due in the next week. The user first chooses a unique identifying name for the task collection. Then the user adds one or more definitions for the collection by selecting any of the available task attributes. Finally, the user decides whether these definitions all need to be true or need only one to be true in order to include a particular task. In other words, both AND or OR operators can be used between these definitions.

User preferences are stored locally where the task manager interface is installed. It is reasonable for a user to keep completely separate and different preference profiles on a work computer as opposed to a home computer. For instance, the task collections shown in the dashboard would evidently be different across different settings. In a work environment, the user would be interested in work related tasks, where as in a personal environment, more personal tasks would be the user interest.

Besides supporting user preferences, TaskPort also provides other useful tools such as the calendar tool. With the calendar tool, the user is able to create customized to-do lists for any particular day or week. Figure 9 presents a list of tasks that are due or need to be started on April 21, 2004. Users can also find out the work density of any month by using the "show density" functionality in the calendar tool. For instance, a very busy day would be marked as red whereas a free day would be white. This is a useful capability that assists users in deciding which days are freer and which are busier. With that valuable information, users can more efficiently spread out their work load and other schedules. Currently, TaskPort employs a naïve algorithm in determining workload. The total density of a particular day is calculated by simply adding up the scores of all tasks that are either due or need to be started on that day. In future implementations, a more sophisticated prioritization scheme could be used to improve the analysis of workload.



Week of 03/21/2004

Due

WebMaster
- Update Personal Website
- Started: 02-20-2004
- Due: 03-23-2004
- Assistant: Unknown

Calendar
- Schedule a meeting with Wei an...
- Started: -
- Due: 03-27-2004
- Assistant: Unknown

Prepare Presentation
- Prepare presentation for senior ...
- Started: 02-27-2004
- Due: 03-27-2004
- Assistant: Unknown

Start

SpacePlanning
- Finding an empty office space
- Started: 03-24-2004
- Due: -
- Assistant: Unknown

End

Figure 9. To-do list organized by whether a task is due or need to be started on a particular day (I'll replace this screenshot with a better one.)

5.5. Achieving scalability

Scalability is a major issue for many user interface application. It applies to not only performance of the application but also effectively representation of large amounts of tasks. Task manager interface is not an exception. Since it is common now for a user to have several thousand emails, the task manager interface must be able to handle several thousand tasks as well. The difficulty here is to represent such large amount of tasks in an efficient and user friendly manner.

TaskPort alleviates the problem of scalability of three different ways. First, TaskPort enables filtering on the existing tasks since users are often only interested in a small subset of tasks at a particular moment. TaskPort implements filtering at two locations. One is the task collection profile in the dashboard view that we have just discussed in the previous section (see Figure 8). The other is the "advanced search" option in the search engine view. Advanced searching function uses the same interface as the setup for a new task collection. The only difference is that instead of display the resulting filtered set as a collection in the dashboard view, it returns the results in search engine.

Second, categorization techniques can be used to further organize results from filtering or directly on all the tasks. To categorize tasks is to organize tasks based on some unique task attribute, such as the task state, the date ranges, or the task names. We already introduced the idea of categorization in our discussion regarding the calendar tool. A customized to-do list (see Figure 9) for a particular day organized by whether the task is due or need to be started can be regarded as first filtered on the task start date or due date then categorized on the action that the user needs to take (whether to start or finish the task). Similar categorization is enabled in the search engine view.

Finally, the concept of having a dashboard and a search engine tool in TaskPort provides a good foundation of dealing with scalability issues. After users define a good profile of task collections to display in the dashboard, they only need to pay attention to the relevant tasks displayed in the dashboard as opposed to all existing tasks. Moreover, having the search engine also allows users to find any tasks from the past.

5.5. Enabling extensibility

Since the development of the task manger interface will be continued in the next few years, we attempt to design TaskPort such that it can be easily extended to include new functionalities and to improve existing functionalities.

As we have discussed in section 5.4, a large portion of TaskPort is based on the ideas of customizable task collections and flexible organization views. Therefore, it is crucial that additional filters, sorters, and organizers can be easily implemented to support this architecture. The following is a list of filters, sorters, and organizers that are currently implemented in the TaskPort interface:

- *NameSorter*: Compares tasks by their names
- *DescriptionSorter*: Compares tasks by their descriptions
- *StartDateSorter*: Compares tasks by their start dates
- *DueDateSorter*: Compares tasks by their due dates
- *ImportanceSorter*: Compares tasks by their relative importance levels
- *NameContainsFilter*: Filter on whether the task's name contains the specified word(s)
- *DescriptionContainsFilter*: Filter on whether the task's description contains the specified word(s)
- *StatusEqualsToFilter*: Filter on whether the task's status is equal to the specified value
- *DueDateIsBeforeFilter*, *DueDateIsAfterFilter*, *DueDateisFilter*: Filters that check whether the task's due date is before or after the specified date, or in the specified week, month, or year
- *StartDateIsBeforeFilter*, *StartDateIsAfterFilter*, *StartDateisFilter*: Filters that check whether the task's start date is before or after the specified date, or in the specified week, month, or year
- *DueDateOrganizer*: Group all tasks according to their due dates (e.g. due today, tomorrow, later this week, later this month, later this year, and other)
- *StatusOrganizer*: Group all tasks according to their status
- *NameOrganizer*: Group all tasks alphabetically according to their names

Any new sorter, filter, or organizer can be easily implemented by following the appropriate interfaces defined in the TaskPort interface. The new functionalities including the corresponding UI components will be integrated into the interface without changing any other parts of the system.

5.5. Enabling integration

Integrating the task manager interface with the rest of the RADAR system is an important step in building a sophisticated cognitive personal assistant such as RADAR. Integration applies both at the communication level and also the design level. TaskPort establishes the foundation for achieving such seamless integration.

From the communications perspective, the current implementation of TaskPort is able to 1) register itself with the task manager backend, 2) create new task in database, 3) update existing tasks by changing any of the task attributes or deleting any tasks, and 4) un-register itself once the user has closed the application. By now, we are very familiar with these basic functionalities. More integration is needed as more components are developed and defined in the future. One example of that is the "obtaining status" functionality mentioned in section 5.2.

Integration at the design level is just as important as that at the communications level. The basic concepts of the task manager interface should align with those of the larger RADAR system. Learning, for instance, is an indispensable component of RADAR. Hence, any task manager interface must also incorporate the ideas of learning in its implementation. Due to the time restriction of this project, current implementation of TaskPort does not directly have learning elements. However, we provided the stepping stones for developing these learning elements in the future. One way learning could be incorporated into TaskPort is that the interface could intelligently observe and learn from user behavior. For instance, if TaskPort learns that the user always seems to expand the hierarchy structure for a particular task at application startup, it may automatically expand the tree for the user in the future. Current implementation of TaskPort has put an internal messaging system in place, where every tree expansion, tree collapsing, task update, task creation and task modification is recorded. This is valuable user behavior data that could be used later to develop a more intelligent task manager interface.

6. Implementation and technical details of TaskPort

The current implementation of TaskPort is developed in Java using Eclipse's SWT (Standard Widget Toolkit) libraries. We made the decision to use SWT as opposed to Java Swing and AWT for the following three reasons. First, SWT contains support for mobile devices such as a PDE. Since the task manager interface will be eventually extended to a ubiquitous environment, portability becomes an important consideration. Second, since SWT uses native widgets in its implementation, any user interface written with SWT gives the user a feeling of a native application across platforms. Third, using the native graphics library also makes SWT faster than Swing.

Figure 10 shows the basic architecture of the TaskPort interface and its connection with the task manager backend.

7. Conclusion and future work

TaskPort serves as a good foundation for creating an effective task manager interface. Ultimately, the goal of our research is to build an intelligent and user friendly task management tool that is fully integrated with the RADAR system and increases user productivity. The following is a summary of how TaskPort takes the first step in creating such task manager interface.

1. Provides users the basic capabilities to browse,

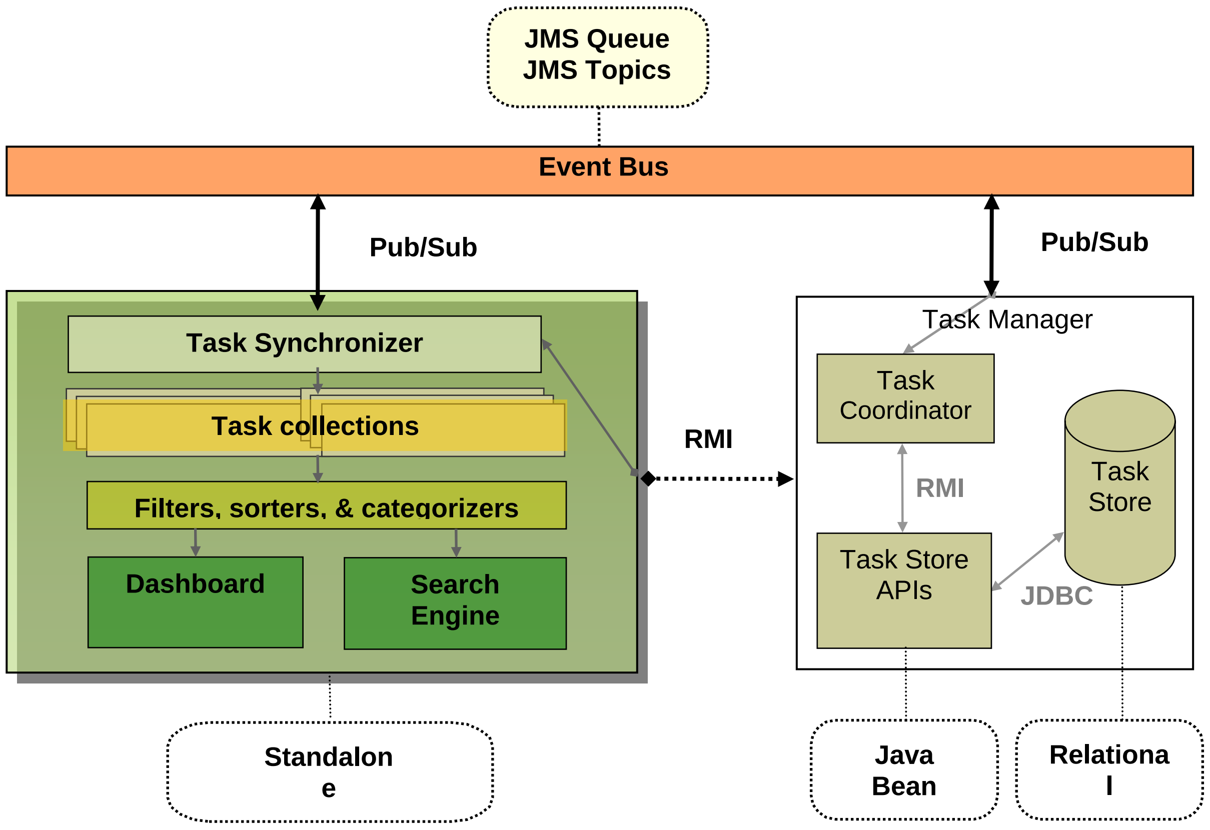


Figure 10 Basic architecture of the TaskPort interface and its connection with the Task Manager backend

modify and create tasks

2. Establishes flexibility at both the protocol level and implementation level. At the protocol level, a hierarchical task structure is defined allowing the support for tasks with varying complexities. At the implementation level, the concept of taskbin collections is put in place to enable categorization of tasks. Moreover, two perspectives of the application: the dashboard view and the search engine view providing both a compact and condense summary of the most relevant tasks as well as a more comprehensive tool that allows searching and categorization on all tasks.
3. Creates usability by being fully aware of user preferences as well as providing other useful tools. Users have the ability to customize how much task detail is displayed, how the color scheme should look, and what collection of taskbin collections should be shown in the dashboard. A calendar tool is capable of not only creating customized to-do lists but also providing an analysis on user workload.
4. Achieves scalability by implementing filtering and categorization on the tasks as well as two perspectives that are appropriate for displaying either small or large set of tasks.
5. Enables integration with the task manager backend via both a Java RMI protocol and a publish / subscribe messaging system that is able to register and un-register the interface and to create, update, and delete tasks.

There is still much room for improvements in TaskPort. Future work can be categorized into protocol, implementation, and evaluation levels. At the protocol level, an even more detailed and flexible task definition could be developed. For example, besides the basic task attributes that we have described, a task could also maintain its full history allowing users to track the progress of the task at both current and future times. Furthermore, in the current implementation, tasks are independently entities. However, in the real word, tasks are inevitably dependent on one another. Therefore, the definition of tasks could also maintain an ordering system that would allow complete workflow management.

At the implementation level, many features in TaskPort can be enhanced or extended. For example, user customization could be extended to other aspects of the interface including the overall look of the application as well as the specifics of how tasks are displayed (not only how much details should be displayed, but how they should displayed). Moreover, learning could be incorporate into the task manager interface making it an intelligent agent. For instance, the interface could learn to deal with failure depending on user actions. In addition, TaskPort needs to be extended to a ubiquitous working environment. In order to achieve that, the interface must first deal with portability and connectivity issues.

At the evaluation level, it is important to conduct a complete user study of TaskPort in order to discover the strengths and weaknesses of the application from the users. Certain features might intuitively appear useful; however a user study could reveal that it may actually hinder the efficiency of the user.

8. Acknowledgements

We would like to thank Professor David Garlan, and Wei Zhang, a graduate student working on the task manager architecture and backend implementation for their ideas, support, and feedback on the project. We would also like to extend our thanks to all members of the Task Manager group.

We would also like to sincerely thank Mark Stehlik for all his help in making this senior thesis possible.

9. References

[1] Bellotti, V., and Smith, I., "Informing the Design of an Information Management System with Iterative Fieldword". Symposium on Designing Interactive Systems 2000. 227-237.

[2] Bellotti, V., Ducheneaut, N., Howard, M., and Smith, I., "Taking Email to Task: The Design and Evaluation of a Task Management Centered Email Tool", In conference proceedings on human factors in computing systems (CHI2003). April 5-10, 2003, Fort Lauderdale, Florida. 345-352.

[3] Bergman, R., Griss, M., and Staelin, C., "A Personal Email Assistant", HPL-2002-236. August 2002.

[4] Kreifelts, T., Hinrichs, E., and Woetzel, G., "Sharing To-Do Lists with a Distributed Task Manager", *ECSCW '93, Proc. Third European Conference on Computer-Supported Cooperative Work*, Kluwer Academic Publishers, Milan, Italy, September 15-17, 1993.

[5] Sousa, J. P., "Task-based Everyday Computing: An Infrastructure", Submitted for publication, April 2004.