International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

THE RADAR ARCHITECTURE FOR PERSONAL COGNITIVE ASSISTANCE

DAVID GARLAN

School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave
Pittsburgh, Pennsylvania, 15213, United States of America
garlan@cs.cmu.edu
http://www.cs.cmu.edu/~garlan

BRADLEY SCHMERL

School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave
Pittsburgh, Pennsylvania, 15213, United States of America
schmerl@cs.cmu.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Current desktop environments provide weak support for carrying out complex user-oriented tasks. Although individual applications are becoming increasingly sophisticated and feature-rich, users must map their high-level goals to the low-level operational vocabulary of applications, and deal with a myriad of routine tasks (such as keeping up with email, keeping calendars and web sites up-to-date, etc.). An alternative vision is that of a personal cognitive assistant. Like a good secretary, such an assistant would help users accomplish their high-level goals, coordinating the use of multiple applications, automatically handling routine tasks, and, most importantly, adapting to the individual needs of a user over time. In this paper we describe the architecture and its implementation for a personal cognitive assistant called RADAR. Key features include (a) extensibility through the use of a plug-in agent architecture (b) transparent integration with legacy applications and data of today's desktop environments, and (c) extensive use of learning so that the environment adapts to the individual user over time.

Keywords: Personal Cognitive Assistant; Agent; Software Architecture

1. Introduction

Computers are playing an increasingly indispensable role in complex day-to-day activities of many people. Email, calendaring systems, daily planners, web sites, and the like are now an essential component of most people's lives.

Unfortunately today's desktop environments provide weak support for carrying out complex user-oriented tasks, or even dealing with the myriad details of handling everyday computer-assisted information, communication, and planning tasks. Although individual applications are becoming increasingly sophisticated, users must map their high-level goals to the vocabulary of specific applications and services, and deal with a barrage of routine tasks, such as keeping up with their email, and keeping their calendars and web sites up-to-date.

An alternative vision is that of a *personal cognitive assistant* (PCA). Like a good secretary, a PCA would help users accomplish their high-level goals, coordinating the use of multiple applications, automatically handling routine tasks, and, most importantly, adapting to the individual needs of a user over time. A PCA would also be able to work cooperatively with the user, automating tasks where appropriate, and staying out of the way where not.

However, realizing such a vision raises a number of hard software engineering challenges. First, to be useful and economical, a PCA should dovetail with existing (and future) applications, file systems, and user processes. Even if one could afford to reengineer all existing desktop applications (which we can't), most users would not be inclined to learn to use an entirely new set of applications, regardless of the benefits provided. Second, a PCA should be extensible. That is, it should be possible to incrementally add new capabilities for personal assistance over time, possibly taking advantage of third-party components to increase the range of support. Third, it should be adaptive. Over time the capabilities of the environment should automatically adapt to the needs and preferences of a user, without a lot of specific user guidance and oversight.

In this paper we describe an architecture and its implementation for a PCA, called RADAR (Reflective Agents with Distributed Adaptive Reasoing), that tackles these challenges head-on. Building on top of existing agent-oriented and distributed systems architectures, RADAR provides a pluggable framework for integrating "specialists" that collectively augment a user's ability to handle complex tasks. Such specialists complement the capabilities of existing desktop environments, applications, and file systems, automating routine (but often complex) tasks programmatically. New specialists can be added or removed at any time. Moreover, learning is a core capability: specialists adapt to the needs and preferences of users over time.

While RADAR is the product of a large number of cooperating researchers, developed over the past three years at Carnegie Mellon University, in this paper we focus specifically on the design of its architecture and the ways in which that architecture supports key engineering properties of compositionality, extensibility, and integration with existing applications and services. We also describe the current implementation and briefly summarize recent empirical results of RADAR's effectiveness in supporting a class of crisis management tasks.

2. Related Work

An important branch of related research is traditional approaches to artificial intelligence, which attempt to automate human-oriented activities such as medical diagnosis, hardware configuration, chess, and robotics. In most of these systems the goal is to have the AI system *replace* the human, and many of these systems have focused on very specific task domains (like chess or medicine). In contrast our work on a personal cognitive assistant attempts to *augment* human capability, and to do this for rather mundane (although often voluminous and complex) tasks like prioritizing email, or helping to manage one's calendar. Additionally, most AI systems have not investigated the engineering issues of developing a component-based approach, or integrating AI capability with legacy systems.

More closely related are other approaches to assisting users with tasks in familiar desktop environments. The Calo project, for example, has been investigating similar approaches.[1] Like RADAR, Calo provides an integration framework for learning-based task-specific components. RADAR differs from Calo in two respects. First, RADAR attempts to co-exist with off-the-shelf applications and data, such as Microsoft Outlook®, while Calo has taken the approach of reengineering standard desktop applications to work smoothly with its task support. The advantage of RADAR is the ability to plug its capability into any desktop environment; the advantage of Calo is that reimplementation of standard applications provides better opportunities for close collaboration between them and the cognitive assistant.

Other work that attempts to help users with ordinary tasks comes out of the ubiquitous computing field.[2,3,4] While these efforts attempt to dovetail with existing infrastructure and applications, their primary focus is on the use of heterogeneous and pervasive devices to help users accomplish tasks more effectively. Work in this area concentrates on configuring and reconfiguring services as users move through an environment. In contrast, RADAR focuses on learning to assist users to do their tasks in a more traditional workplace environment.

Another closely related area is that of agent-oriented architectures.[5,6,7,8,9,10] Over the past decade there has been considerable interest in multi-agent systems and middleware to support them. In particular, a number of architectural frameworks have been proposed, including AAS,[11] Zeus,[12] and FIPA..[13] As described later, we build on top of agent-oriented architectures (and, in particular, FIPA), specializing the general notions of agents and agent coordination with the specific architectural structures that characterize the RADAR architecture.

There are many applications that help users manage their daily tasks; most are limited to managing todo lists, and reminders; some of them have intelligence built in. The Personal Information Management workshop,[14] for example, contains examples of systems that integrate with email and help manage other personal information. The vision of Radar is broader than these systems in that it encompasses personalized learning and collaboration between people to help automate tasks. Another example is the integration between Google Mail and Google Calendar; if Google Calendar recognizes some text in an email message as a calendar event, it will provide a button to automatically add the event to the Google calendar. While this task entails some intelligence, it does not help with scheduling the meeting, coordinating with other people or the other issues associated end-to-end with meeting assistance. Nor does it learn the idiosyncrasies and preferences of when someone would like to meet.

3. Architectural Requirements for a PCA

The vision of a PCA is that of a smart assistant, that in some sense "understands" the user, helping out where needed and effective, but staying out of the way otherwise. Inherent in this view is the idea that a PCA should complement what a user normally does, and how a user normally does it. Although over time a user might adapt her behavior to rely more heavily on the PCA as she gains trust in it, she should not be forced to do this. In particular, there should be a level of user-machine coordination and collaboration.

Consider the following scenario: A busy user has loaded a PCA onto his desktop. At first the user notices little change to his normal way of working. However, exploring the PCA console, he discovers that he can activate a calendar assistant. After activating it, the user is prompted to identify some general preferences for things like what calendaring application he wants to use, what times to keep free on the schedule, cancellation policies, and the like. Since the user is wary of turning over control to any automated calendaring assistant, he decides to be conservative, requesting that the assistant should schedule meetings only during the hours of 10-12 on weekdays, always confirming schedule changes before committing them, and never canceling or rescheduling an existing meeting. As time progresses he notices that the calendaring assistant has been able to correctly identify email messages that relate to scheduling requests, and to suggest reasonable scheduling actions. Based on positive experience, he decides to let the assistant do it automatically. Over time, he discovers that the assistant can do more and more: it learns his desires for canceling meetings (e.g., preferring to move subordinates' meetings before those with his boss); it learns that when the user goes on vacation or business travel, email should be sent to people with whom he has regular meetings to let them know, etc. Quite happy with this capability he continues to let it do more, confident that it is learning how he would like it to be done, and asking for permission before attempting anything radically new.

From an engineering perspective this vision implies three essential requirements for a PCA:

(i) **Compatibility:** The services and assistance provided by a PCA should co-exist with the capabilities of current legacy applications and services. The user should not have to abandon old ways of doing business, or learn to use new applications with different interfaces. While additional capability provided by a PCA will necessarily require some additional forms of user interaction, these should supplement, not replace, existing forms of interaction. This implies compatibility not only with applications, but also with information sources as well. For example, email messages are often an important stimulus and information source for an assistant (for example, signifying the need to start a new task). Understanding email messages, written in a natural language, and stored in standard email repositories (e.g., Imap), is essential.

(ii) **Extensibility:** It should be possible to incrementally augment the capabilities of the PCA. For example, if some new form of task assistance becomes available, it should be easily pluggable into the existing system, adding new capability without disrupting the old, and dovetailing with existing assistance provided by the PCA. One can even imagine a marketplace for personal task assistance in which different forms of the same kind of assistance might be purchased at different price-quality points.

(iii) **Adaptability:** The system should conform to the user, learning new opportunities for assisting the user, and inferring appropriate behavior based on how users carry out their tasks. Learning should apply to a wide variety things, including prioritization of tasks (e.g., helping a user focus on the important things), policies for interaction with others (e.g., deciding who should have access to certain kinds of information), clusterings of related activities (e.g., noticing that if action A is performed, action B is usually also performed), interpretation of natural language (e.g., recognizing idioms that relate to task achievement), and many others.

In addition to these requirements, there are a number of other more-standard systems-oriented engineering qualities, such as robustness, availability, security, and performance. Indeed, the services provided by the PCA should have comparable quality attributes to today's mail systems, which tend to be available in a global setting, highly robust, secure, and reasonably efficient.

4. The RADAR Architecture

To achieve these goals, RADAR has adopted the run-time architecture pictured in Fig. 1. In this figure, the architecture is depicted from the point of view of a single individual working in a personal RADAR space. We first give a high-level overview, and then look in more detail at specific technical issues.

4.1. *Overview*

At the bottom layer are legacy applications, services, and data stores. Applications include things like email readers, web browsers, calendar managers, and the like. Data stores include documents stored in local and remote file systems, repositories of email, calendar information, contact lists, etc. Users interact with these in normal ways. The challenge is how to integrate with existing applications. To achieve this, application APIs are used or written. For example, Monitor (M) interfaces inform RADAR of events that happen in an application (e.g., the user moves an appointment), that might trigger new tasks or learning by RADAR; Control (C) interfaces allow RADAR to make changes to the desktop space (e.g., to schedule a new appointment); User Interface embellishments (UI) allow RADAR to present information to the user in a manner the user is familiar with (for example, to display RADAR-proposed alternatives for a meeting on a user's calendar). In addition we add a RADAR Console, which provides a user with direct access to RADAR and its capabilities. In many modern applications, these kinds of interfaces are possible through standard APIs, such as .NET. When these aren't available, the information is usually accessible indirectly through manipulating the data that the applications use.

On top of this layer, RADAR adds a layer of task assistance. This layer is composed of five types of components:

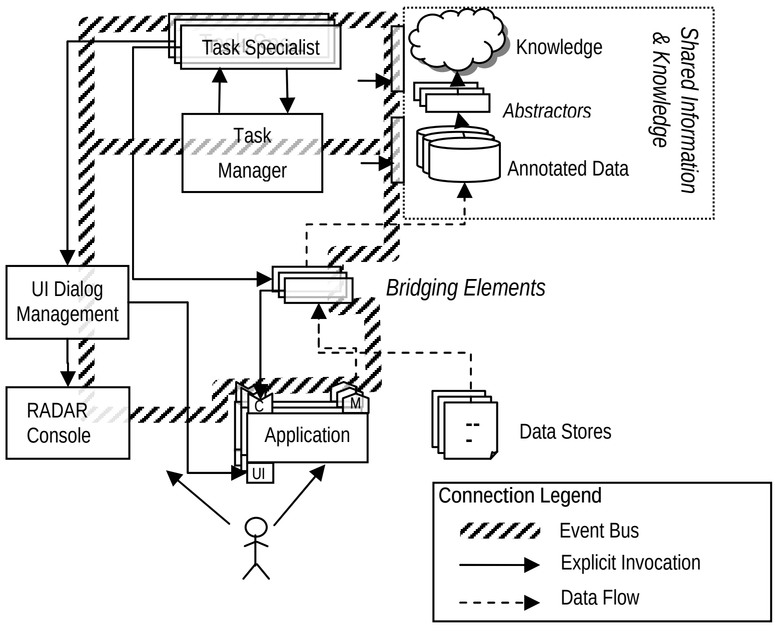


Fig. 1. The Architecture of an Individual's Radar.

Task specialists: A *task specialist* (or just *specialist*) is a component that attempts to provide assistance for a particular kind of task, such as schedule management, web site updating, and routine email handling. The number and kind of specialists can vary from user to user, and over time for a single user as new specialists are added or removed from that user's RADAR space. Each specialist contains knowledge about how to conduct a particular task, and each contains a learning component that allows the specialist to adapt to the user with respect to preferences, preferred methods of doing the task, etc. Specialists may store their learned knowledge in the shared knowledge base (see below), which is accessible to other components. In this way, knowledge can be shared and transferred to other specialists. For example, if I consistently defer a meeting with Victor when there is a conflict, the scheduling specialist will learn that dealing with Victor is not high on my list of preferences. The specialist involved in organizing my email may also use this information to sort my email appropriately.

Task management: To coordinate the work of the specialists and to provide overall tracking and control of tasks is a *Task Manager*. The Task Manager comprises a number of logical services, including task dispatch (interacting with specialists to assign new tasks), task tracking (keeping track of high-level state of tasks – see below), task query (retrieving all tasks that match certain selection criteria), and task prioritization (keeping track of the relative priority of tasks). In addition to this, the Task Management component is responsible for data security and privacy.

Shared information and knowledge: To be effective, specialists and task management services must manipulate data in richer forms than is conventionally stored in today's desktop environments. For example, intelligent email assistance requires that key features of email messages are identified and classified. Similarly, calendaring information may need to be structured in higher-level ways than is natively stored by a calendar system. In addition, there is the need to represent knowledge between high-level entities and relationships in the user's world. For example, social nets that contain knowledge of the relationships that a user has to other people, are stored in a knowledge base; social nets can be used by specialists and task management to determine security procedures (e.g., who are my friends), policies for actions (e.g. don't cancel a meeting scheduled by my boss), and general knowledge about the environment (e.g., what rooms are physically close to my office).

Bridging elements: To get information from the desktop level into RADAR space, requires certain bridging elements. There are several kinds of these. One kind transfers information from desktop space into RADAR space. These include categorizers and extractors that understand natural language to label and categorize the information from the desktop space. A second kind of bridging element takes information directly from legacy applications, through the M interface in Fig. 1. These allow RADAR specialists to monitor activities performed directly by legacy applications, and to control those applications programmatically. The difference between such bridging elements and specialists is that, although they may both have knowledge specific to particular tasks, bridging elements are responsible for transforming native representations of data (such as textual email) into task-oriented information (such as the existence of a new task), while specialists have knowledge of how to assist the user in *carrying out* the tasks.

Connectors: In order for components within an individual's RADAR to interact, we have defined specific protocols to make explicit the roles and nature of interaction between different components. There are three types of connectors in RADAR:

(i) *Explicit Invocation* connectors are used when synchronous requests are made between components, and the target component is known to the component issuing the request. Furthermore, information may be returned along these channels. Such connectors are used when components issue commands directly to other components, and need to know that the commands have been met before they can continue. For example, a specialist may request that the Task Manager update the state of a particular task; it should not continue interacting with the task until it has received a reply about the success of that interaction.

(ii) An *Event Bus* connector is be used to facilitate learning by informing subscribers about events that occur in the system. At a minimum, whenever a command is issued, a concomitant event is announced. In addition, components can freely announce their own events. For example, a component that learns social net information would be expected to announce an event when it has learned that Melinda is John's boss. Every component in RADAR is allowed to publish and subscribe to events.

(iii) *Data flow* connectors show the passage of data through RADAR. For example, the bridging elements take data from the Data Stores, apply natural language extraction, and produce Data in the annotated database that contains markers to interesting features in the Data Stores (for example, where names occur in email messages).

User Interaction: One important requirement of the flow of information through RADAR is the need to manage interaction with a user of RADAR. If a part of RADAR wishes to communicate with the user, it should only do so at appropriate times. For example, a specialist may want the user to confirm some information. If it immediately interrupts the user to request this, it might distract the user from working on another task, causing him to lose context. For real world use, this will most likely make the user less efficient because he is constantly being interrupted. Thus, all Radar-initiated interaction with the user is mediated through the UI Dialog Management component, which manages when and how a user should be interrupted. The UI Dialog Manager learns when and whether to interrupt the user,[15] based on knowledge of the user's focus and interruption policies. The UI Dialog Manager might present this information via the RADAR console, by RADAR-specific UI embellishments in legacy applications (using the UI interface in Fig. 1), or other interfaces using techniques similar to those described by Faulring and Myers.[16]

Fig. 1 shows only the architecture of an individual's RADAR. However, an individual's RADAR must communicate with others, both those that use RADAR and those that do not. To ensure privacy and access control, the Task Manager mediates RADAR's communication with other users. This gives RADAR the opportunity to also determine how best to communicate with a particular person. For example, if that person has their own instance of RADAR, then this component can communicate with their RADAR using message passing; if the person does not use RADAR, it will choose alternative ways to communicate (e.g., email, IM, cell-phone text message).

4.2. *Information Flow*

To illustrate how information from the desktop space flows through RADAR, consider the arrival at John's desktop of an email from `fred@a.com` containing the text "I would like to organize a meeting with you and Melinda next Tuesday." Furthermore, the email contains details of Freds' free times on Tuesday.

(i) Categorizers and extractors take this information and annotate it with structural information such as the existence and location of names ("Melinda"), dates ("next Tuesday"), and that the message is concerned with organizing a meeting. A new task for organizing a meeting is also constructed by the extractor and sent to the Task Manager for dispatch. The task is stored in a traditional database (as annotated data in Fig. 1) and contains a reference to the original message.

(ii) Abstractors take the structured information and further annotate it with knowledge. For example, an abstractor notes that Melinda is John's boss, and that John prefers meetings on Tuesday to be in the morning. This knowledge has been learned previously based on John's previous interactions. The knowledge is stored in a knowledge base, which can be queried by other components.

(iii) The Task Management component notices that a new task to organize a meeting has been proposed, via triggers in the task database. Based on John's policies and previous behavior, RADAR determines that it is permitted to automatically schedule the meeting. It identifies a task specialist that is responsible for managing John's calendar, and assigns the task to it. Meanwhile, the console is informed through event notification that this task is being handled by RADAR, and adds the task to its list of tasks so that, if interested, John can find out what RADAR is currently doing.
(iv) The Task Specialist attempts to find suitable slots on John's calendar for the meeting to take place. This might involve confirmation with John, which will be done through UI dialog management.
(v) The specialist will need to collaborate with Melinda to determine a final meeting time. The specialist submits a request to find Melinda's free time on Tuesday to the Task Manager. The Task Manager determines that Melinda uses RADAR, and communicates with her RADAR to get her free times. (Melinda's RADAR receives the request as a task and processes it in a similar manner.) When the specialist receives Melinda's free times on Tuesday, it chooses a shared free time and schedules the meeting on John's and Melinda's calendars.
(vi) RADAR sends mail to Fred indicating the time when the meeting has been scheduled.

4.3. *Task Management*

A central notion of RADAR is the idea of a *task.* A task is a unit of work that can be automated (or partially automated) by RADAR. The unit of work could be assigned to a single task specialist, or it may involve the coordination (through a task planner) of multiple task specialists. Such a planner would itself be implemented as a specialist.[a]

A key component in managing tasks in RADAR is the Task Management facility, which is responsible for the following task-related duties.

4.3.1. *Task Dispatch and Specialist Registry.*

The Task Manager acts as a directory facility for matching particular types of tasks to specialists that can be used to automate them. The Task Manager is then responsible for assigning tasks to specialists, and also indicates to specialists when to suspend or stop particular tasks (for example, at the user's behest, because the task is no longer valid, or another task is more important). In addition to dispatching tasks, this component is also responsible for detecting the liveness and availability of particular specialists.

4.3.2. *Task Data Management.*

The Task Data Manager manages changes to a task. Once a task is assigned to a specialist, a specialist can only make changes to the task through the task data manager. In addi-

[a] The current implementation contains only a rudimentary planner. See also Section 7 on future work.

tion to ensuring that the changes to the task are legal, the task data manager is responsible for (1) persisting the changes to the task; (2) announcing changes to a task; and (3) keeping a history of the changes to a task. These facilities enable learning about tasks and help with later user behavior studies and offline learning.

Table 1 describes the fields associated with a task at the RADAR level. The Task Manager deals with information about tasks at a level that is common to all tasks, and that aid in task management and informing the user about what RADAR is doing. Task-specific information is stored by other components in the contents field a task, and this information is treated as a black-box to the Task Manager.

Table 1. The fields associated with a RADAR Task.

Field	**Type**	**Description**
id	UUID	A universally unique identifier to identify the task. This is generated by RADAR.
state	enum	An indication of the state that RADAR is in. The states and state transitions for a task are described in Table 2.
type	string	Used by RADAR to choose which specialist to assign the task to.
relatedTasks	Map	Allows access to related tasks. The key to the map is the type of relationship, and a collection of task identifiers is returned. For example, using the key SUBTASKS, all subtasks of a task may be retrieved.
created, due, started ended, updated	date	Information about various important times associated with a task. For example, the updated field reflects the most recent time that any update occurred to the task.
contents	object	The specialist-specific contents of the task. This object stores internal state and information about the task, that is treated as opaque to the Task Manager.
description	string	A human-readable short description of the task that can be displayed to the user to indicate what the task is about.
progress	string	A human-readable short description of the progress of the task, as set by specialists. For example, the progress for the task in Section 4.2 might indicate that RADAR has sent available times to Fred.
originator	person	The person who caused this task to come into being. For example, in Section 4.2, Fred will be the originator.
sources	collection	A link to the annotated data that contains the source of this task. For example, the email message requesting that a meeting be arranged.

Tasks also have a high level state machine that defines the legal transitions that a task may go through. The transitions and their descriptions are described in Table 2. As the tasks are taken through their paces, the states provide a quick way of letting the user know what is happening. Specialists may have their own internal states when they are running the tasks, and these should be presented to the user in human readable form as progress associated with the task.

When a task is first extracted from email, it is placed in the CREATED state. The contents of the task contain the form that was extracted from the contents of the email, with some of the fields filled out based on natural language understanding. The user must then be satisfied that the form has been filled out correctly. The policy for this can be set by the user; the user can examine the form and make corrections, or trust the extractors to have done a good job, or by specifying the extraction confidence level for which the user is happy. The task is then progressed to the READY state. In this state, it is assumed that all the information necessary for a specialist to start working on the task is correctly in the contents of the task. RADAR can then assign the task to a specialist; again, this can be done automatically by RADAR or manually by the user, depending on policies. When the specialist starts working on a task, it sets its state to RUNNING. From this state, the task can either complete, fail, the user may suspend the task, or the specialist can report an error. The ERROR state indicates that the specialist could not understand the content of the task, and so the user is prompted through the UI Dialog Manager to correct the contents.

It is also possible that RADAR could be stopped (e.g., for maintenance) and restarted. In this case, the Task Manager has support for putting tasks in a HIBERNATED state; it communicates with specialists to ensure that the contents of tasks are in state where the specialists can continue working on them later. When RADAR is restarted, any tasks that are HIBERNATED are reactivated and reassigned to specialists.

Table 2. The state transitions allowed for Tasks.

State	CREATED	READY	ASSIGNED	RUNNING	SUSPENDED	CANCELLED	COMPLETED	FAILED	DELETED	ERROR	Description
CREATED		X							X		The task has been extracted from a data source.
READY			X						X		All information about the task is correct and can be processed by RADAR.
ASSIGNED				X						X	The task has been assigned to a specialist, but the specialist is not yet doing the task.
RUNNING					X	X	X	X	X	X	The specialist is in the process of executing the task.
SUSPENDED				X							The task has been suspended by RADAR.
CANCELLED											The task has been cancelled.
COMPLETED											The task completed successfully.

FAILED											The task was unable to complete successfully.
DELETED											The task has been deleted.
ERROR		X									There is an error associated with the task that the specialist cannot understand. It requires input from the user.
HIBERNATED			X	X							The task is in hibernation, when RADAR was stopped.

4.3.3. *Information privacy and access control.*

In many instances, users of RADAR will want to restrict information that is made available to others. For example, a user may not want to make calendar details available to others, and may not want RADAR to automatically schedule meetings if they are requested from certain people. While the knowledge particular to this lives in the shared knowledge base, a part of the Task Management facility is responsible for ensuring that the user's preferences are met.

A *Security Manager* is the part of the Task Manager that maintains privacy and security protection in RADAR. At startup, the Security Manager provides authentication for both users and RADAR components. The Security Manager provides a basis for applying message encryption-decryption within RADAR. By performing mutual authentication between two entities in RADAR, both entities obtain each other's credentials (e.g., public keys) used to encrypt and decrypt communications between the pair of entities based on a supported encryption mechanism (e.g., public-private keys encryption).

When a user or specialists belonging to the user tries to access some information or resource, the Security Manager consults a *Policy Manager* to determine whether the user is authorized to access the information/resources or not. Once the access rights have been established, the Security Manager acknowledges and publishes the access rights status so that status can be shown to the user.

When policies are changed on the fly, either due to users or due to changes in system environment, the Policy Manager will inform the Security Manager of the changes and the Security Manager will broadcast the policy changes. Specialists running tasks that rely on the policy should then acknowledge the change and modify their behavior to reflect the new policy.

Because the Security Manager is part of the task management infrastructure, the Task Manager becomes the arbiter for all actions relating to a task. It is intended that, in the future, the Task Manager will also filter information to maintain required privacy.

The main responsibilities of the *Policy Manager* are to handle access control policies, both current and revoked, that are stored in the knowledge base. When the Policy Manager receives a query from the Security Manager, it checks whether the queried task violates any current access control policies or not, and sends the results back to the Security Manager.

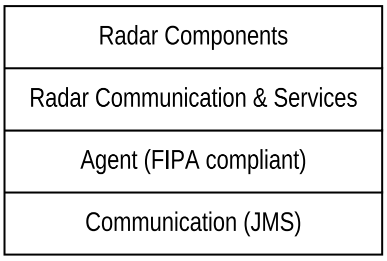


Fig. 2. RADAR Layered Implementation Architecture.

The Policy Manager also provides a front-end interface for defining access control policies via the RADAR Console. It tells the user the kinds of policies that can be defined (using information about social networks in the knowledge base) and notifies the user about policy changes when applicable.

When the user make changes to existing policies or when there is a change in the knowledge base that alters the existing policies, the Policy Manager will be triggered and a list of affected policies will then be sent to the Security Manager.

5. Implementation

RADAR is designed to run as a server-oriented system in which the main capabilities are provided in stable environments that communicate with a user's personal desktop or mobile platform. As such, RADAR task management and assistance operates much like email servers, communicating with mail clients, but accessing mail stored in a stable way on externally-maintained and robust servers. This design helps provide the needed availability required to support a continuous, globally accessible service.[b]

The implementation of RADAR is based on a layered use of existing technology, illustrated in Fig. 2. At the lowest implementation layer are standard middleware services for distributed systems. Specifically, we use Java Messaging Services (JMS), which provide a network-wide service for sending messages between components. The interface to this layer provides an API that hides details of the middleware, supporting basic communication mechanisms for remote method invocation and publish-subscribe.

At the next higher level is an agent-oriented architecture, which provides a virtual agent layer. The agent layer provides a FIPA-compliant API that defines the types of messages that can be used to exchange information between components, and specifies the building blocks on which more sophisticated communication protocols are built.

The RADAR Communication and Services layer specializes more general agent-oriented paradigms, defining specific protocols for communication between specialists

[b] Although targeted for server-oriented deployment, RADAR also permits client-oriented configurations in which more of the functions run on the client side.

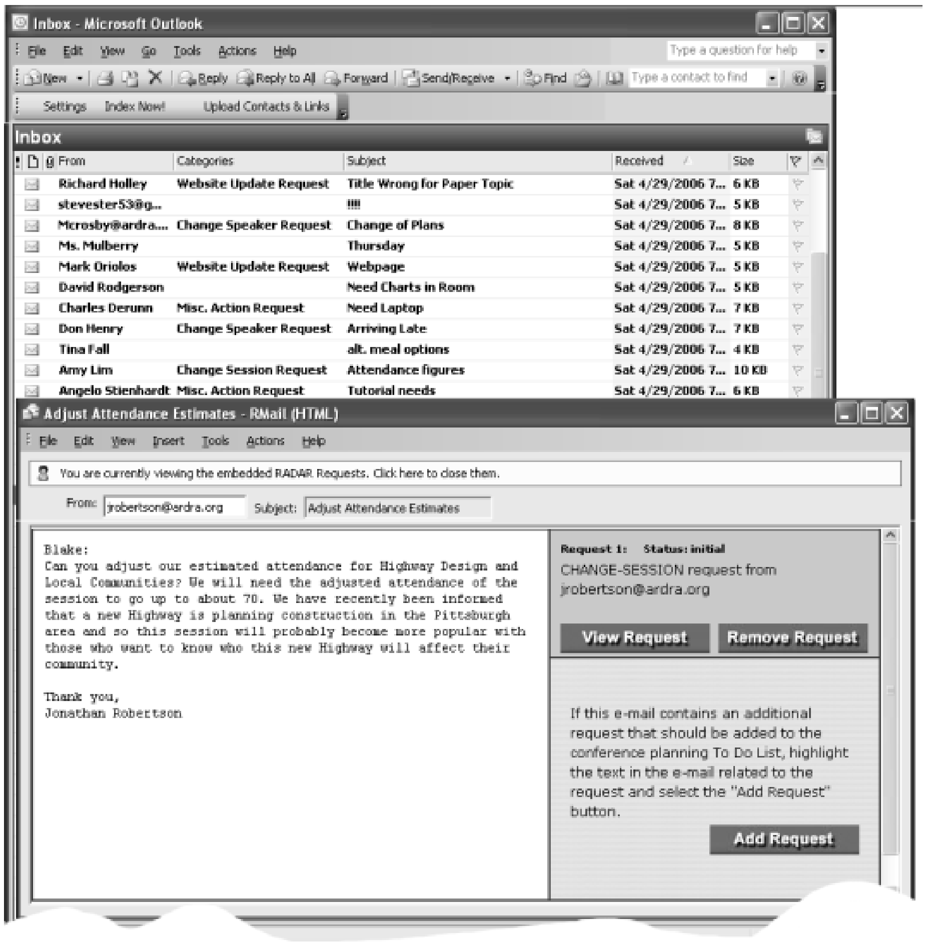
and the task management services, interaction with the knowledge base, registration and invocation of the bridging elements. This layer defines the rights and responsibilities for specialists, bridging elements, shared data and knowledge through a set of interface specifications. In addition, common RADAR Services are provided; these include the console, Task Manager, UI Dialog Management, and communication services that provide interaction between RADARs.

Building on top of this architectural infrastructure, RADAR V1.0 includes the following components and capabilities:

- Extractors and categorizers that understand general language terms such as places and names, as well as task-specific information such as scheduling constraint requests.
- Specialists that assist the user with:
  - Managing a company website, by correcting errors in people's information based on emails, and publishing the updates to a website; [17]
  - Managing a schedule, which includes scheduling appointments and finding spaces where meetings can take place;
  - Preparing work summaries, or briefings, that can be sent to superiors, by learning which emails and tasks are more important and helping the user to summarize this information;[18]
- The Scone knowledge base,[19] which stores and infers knowledge learned by various components of RADAR. In this RADAR V1.0, Scone was used primarily for storing and inferring information learned about email messages.
- Integration with Microsoft Outlook®, for organizing users' email and as a user interface for controlling some aspects of Radar. For this, Outlook's COM interface was used to support the UI, C, and M interfaces (of Fig. 1), providing natural extension points from which to integrate Outlook with RADAR. **Error! Reference source not found.** provides a screenshot of this integration. The window in the background shows the Outlook inbox list. RADAR has added a column that indicates the type of task that a particular email corresponds to. When an email is opened, RADAR inserts requests (tasks) that it has understood to the right of the email message, and allows the user to add additional requests that might be associated with the email.

6. Evaluation of RADAR in Use

While designed to promote the requirements outlined in Section 3, a critical question is how well RADAR performs in a live setting, and how effective is learning in automating everyday tasks. To investigate these questions, the RADAR team carried out extensive experimental evaluation. This evaluation was developed and conducted by other members of the RADAR project, who have published the details of the results.[21] We summarize the results here so that readers will have a frame of reference for our efforts.


Inbox - Microsoft Outlook
File Edit View Go Tools Actions Help
Type a question for help
New Reply Reply to All Forward Send/Receive Find
Type a contact to find
Settings Index Now! Upload Contacts & Links
Inbox

From	Categories	Subject	Received	Size
Richard Holley	Website Update Request	Title Wrong for Paper Topic	Sat 4/29/2006 7...	6 KB
stevester53@g...		!!!!	Sat 4/29/2006 7...	5 KB
Mcrosby@ardra....	Change Speaker Request	Change of Plans	Sat 4/29/2006 7...	8 KB
Ms. Mulberry		Thursday	Sat 4/29/2006 7...	5 KB
Mark Oriolos	Website Update Request	Webpage	Sat 4/29/2006 7...	5 KB
David Rodgerson		Need Charts in Room	Sat 4/29/2006 7...	5 KB
Charles Derunn	Misc. Action Request	Need Laptop	Sat 4/29/2006 7...	7 KB
Don Henry	Change Speaker Request	Arriving Late	Sat 4/29/2006 7...	7 KB
Tina Fall		alt. meal options	Sat 4/29/2006 7...	4 KB
Amy Lim	Change Session Request	Attendance figures	Sat 4/29/2006 7...	10 KB
Angelo Stienhardt	Misc. Action Request	Tutorial needs	Sat 4/29/2006 7...	6 KB

Adjust Attendance Estimates - RMail (HTML)
File Edit View Insert Tools Actions Help

You are currently viewing the embedded RADAR Requests. Click here to close them.

From: jrobertson@ardra.org Subject: Adjust Attendance Estimates

```
Blake:
Can you adjust our estimated attendance for Highway Design and
Local Communities? We will need the adjusted attendance of the
session to go up to about 70. We have recently been informed
that a new Highway is planning construction in the Pittsburgh
area and so this session will probably become more popular with
those who want to know who this new Highway will affect their
community.

Thank you,
Jonathan Robertson
```

Request 1: Status: initial
CHANGE-SESSION request from jrobertson@ardra.org

If this e-mail contains an additional request that should be added to the conference planning To Do List, highlight the text in the e-mail related to the request and select the "Add Request" button.

Fig. 3. The RADAR Console UI, integrated with Microsoft Outlook®.

A controlled crisis scenario was constructed: a week before a conference is due to start, a building that was to be used to host the conference becomes partially unavailable. Subjects in the experiment were asked to reschedule the conference sessions in alternative rooms, manage the constraints on speakers who have already booked travel assuming the previous scheduled, brief the program committee on progress, and stay current with arriving email. The crisis is exacerbated by the fact that the primary conference organizer is unavailable to help, although he used RADAR to help organize the conference initially.[c]

[c] This test will be conducted annually for the duration of our current funding, so that improvements and additions to the RADAR technologies can be tracked, including the work reported in this paper. We report here on the results of the test conducted in May 2006, which used a version of the architecture that was significantly tailored to the specific components used in the crisis.

Two instantiations of RADAR were used in the experiment:

(i) *Without any information learned about the conference.* This tested the effect of RADAR without it having prior specialization to crisis situation. It does not know, for example, whether a particular message concerns a conference event. There were 31 subjects in this group.

(ii) *With preloaded knowledge* learned as if RADAR had been used by the conference organizer to organize the conference initially. The various learning components had been trained using the RADAR user interface under regular task activity. This group contained 32 subjects.

In addition to these configurations of RADAR, a control situation was also run in which participants used only off-the-shelf tools.

In all cases, test subjects were given two hours to work the conference planning crisis scenario. In this test, learning was shown to have statistically significant positive influences on several machine learning system performance metrics, and that both uses of RADAR were statistically better than the control situation. This means that on average, users of RADAR were more adept at handling the crisis in the test, and that RADAR, when trained, allows users to perform statistically better than when RADAR has not been trained.

7. Discussion

The architectural design of RADAR addresses the three critical requirements for a PCA outlined in Section 3. First, *compatibility* is supported through the layered architecture, which augments existing applications and data without replacing them. While applications must be modified in small ways to provide monitoring and control capabilities from the RADAR layer, and have certain user interface enhancements, by and large they remain unchanged. While the interface to each legacy application will differ, our experience in another project[20] suggests that wrapping applications to provide the necessary interfaces is possible, and is becoming increasingly easy with modern applications. We are, however, limited to facilities provided by the interfaces of applications.

Second, *extensibility* is supported through a component-oriented architecture in which task assistance is provided by modules (specialists) that can be incrementally added to or removed from the RADAR ensemble, simply by registering or deregistering them.

Third, *adaptability* is supported in several ways. The RADAR console allows a user to specify policies directly. In addition each specialist and the Task Manager provides its own learning capabilities, as outlined above, which coupled with a shared knowledge base, and common mechanisms for learning (e.g., extractors and abstractors) allow RADAR to adapt over time to a user's needs.

However, our experience with Radar in the test has indicated several areas of improvement that are needed in the architecture, to increase its flexibility. The first is the way in which extractors and categorizers hoist information in the RADAR system. In the original architecture, this was done in a coordinated fashion, relying on a designated component to coordinate which extractors were called based on information from the categorizers. In reality, there are several general-purpose extractors (for example, to extract names, addresses, etc.) that can be used on any message. In the next iteration of the architecture, we believe that a blackboard style of interaction between categorizers and extractors is required.

The second area of improvement relates to the need to support different modes of user interaction. Our original architecture assumed that categorizers and extractors would perform well enough to understand a task from an input to RADAR, and that the user would only need to be involved once a task had been defined. Other teams in RADAR were less optimistic about this, and so required a step in RADAR that required the user to confirm the extraction process, and make any necessary adjustments to the understanding of the task before it comes under the purview of task management. In fact, this is essentially a trust issue that we had not considered in the architectural design – over time, RADAR should learn to extract information more accurately, and the user should be able to trust that RADAR can elicit tasks from the message more accurately; users should therefore have control over how much interaction they have with RADAR. To achieve this, we have required that the extractors provide information about the confidence that extraction is accurate, and that this information is passed to the Task Manager when a task enters RADAR. Users can specify a threshold below which they require RADAR to ask them to confirm. This part of the architecture will evolve into a more sophisticated element that implements trust management.

Third, the decision to separate extractors from specialists has met resistance from some implementers of specialists. We believe that this is mainly due to the fact that the teams responsible for writing the knowledge in the specialist for performing a particular task are in many cases the same team writing extractors that can understand the information about their tasks from, for example, email. In this case, the teams have been tempted to circumvent the architecture and have the specialists and extractors directly communicate and share information. We believe, however, that the separation of concerns at the architectural level is still the right design, but we are investigating the best interaction models that will allow sharing of information between specialists and extractors in a more principled way.

Finally, our initial implementation of the RADAR architecture provided a FIPA compliant agent communication layer. We provided an additional specialization of this layer to provide RADAR communication protocols that were designed for RADAR, and take into account the requirements outlined in this paper. We plan to evaluate whether other interaction models (e.g., service oriented architectures) would be a better model for implementing the agent communication layer between services in RADAR, given our experience with providing existing RADAR services. One of the strengths of the layered approach is experiments using different underlying implementations should have minimal impact on the existing RADAR services.

8. Conclusions and Future Work

Realizing the vision of a fully-featured PCA is a formidable task that will take significant advances in research and engineering to achieve and demonstrate. In this paper we describe first steps toward realizing that vision. The key to this is the design of a pluggable architecture that permits extensibility and adaptability, while remaining compatible with existing desktop services and applications. The implementation of RADAR v1.0 and its performance on tests are encouraging: it demonstrates that an integrated task management system can be implemented and be effective even in handling highly-stressful situations and with complex tasks.

However, considerable work remains to be done to fully realize the potential of a PCA. First, are the architectural changes that we have mentioned in the previous section. Second, is the discovery of new forms of learning that can help the user. With respect to the crucial capability for learning to provide better task management, for example, we are now exploring the possibility of learning such things as how to order tasks according to their importance. In particular, we think it should be possible to take into consideration such things as the type of task, the history about how quickly similar tasks have been completed, and who originated the task, to predict the importance of task when it enters the system.

Third, is representation and assistance with complex tasks, for example those that contain workflow involving multiple specialists In many cases such tasks will require planning as well as learning. This requires research on combining learning and planning in complex tasks, as well as implementation mechanisms to make such capabilities available as common services to RADAR specialists.

Fourth, is the need to provide a user with greater transparency into the workings of RADAR. While RADAR can provide demonstrable value-added to users, at present it is unable to explain its actions in a way that allows the user to understand exactly what RADAR has learned and why it believes what it does. Partly this is due to the nature of statistical machine learning, but it also related to the enhancement of specialists so that as a part of their normal functionality they can explain their task understanding and actions in user-oriented terms.

Finally, we expect that with the incorporation of more learning components, the knowledge base will be more fully utilized. Currently, the knowledge base is used by extractors to aid in understanding natural language, and also in defining access policies. We anticipate that the knowledge base will participate in task planning and strategy, and in filling out forms by better inferring the identity and roles of people are that send messages to a user's RADAR.

Acknowledgements

Many people at CMU have participated in building RADAR and running the tests, and we would like to acknowledge them for their hard work. In particular, we would like to thank Aaron Steinfeld, who helped to make our evaluation section accurate, and the development team for the RADAR architecture (Jaime Oviedo, Gabriel Zenarosa, Nicholas Sherman, and Pongsin Poosankam).

This material is based upon work funded by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

References

1. Berry, P., Myers, K., Uribe, T., and Yorke-Smith, N. Task Management under Change and Uncertainty. Constraint Solving Experience with the CALO Project. Proc. CP'05 Workshop on Constraint Solving under Change and Uncertainty. Spain, 2005.
2. D. Avrahami and S. Hudson, QnA: Augmenting an instant messaging client to balance user responsiveness and performance. *Proceeings of the ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 515-518, Jan. 2004.*
3. J.P. Sousa, Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments. Ph.D. Thesis, Carnegie Mellon University School of Computer Science Technical Report CMU-CS-05-123, 2005.
4. Want, R.; Pering, T.; Danneels, G.; Kumar, M; Sundar, M.; and Light, J., "*The Personal Server: changing the way we think about ubiquitous computing*", Proc. of Ubicomp 2002: 4th International Conference on Ubiquitous Computing, Springer LNCS 2498, Goteborg, Sweden, 2002.
5. S. Cranefield, M. Purvis, An agent-based architecture for software tool coordination, in *the proceedings of the workshop on theoretical and practical foundations of intelligent agents*, Springer, 1996.
6. T. Finin, J. Weber, G. Wiederhold, et al., Specification of the KQML Agent-Communication Language, 1993.
7. S. Franklin, A. Graesser, Is it an Agent or just a Program? A Taxonomy for Autonomous Agents, in: *Proceedings of the Third International Workshop on Agents Theories, Architectures, and Languages*, Springer-Verlag, 1996.
8. M. R. Genesereth, S. P. Ketchpel, Software Agents, Communications of the ACM, Vol. 37, No. 7, July 1994.
9. B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, M. Balabanovic, A domain-specific Software Architecture for adaptive intelligent systems, IEEE Transactions on Software Engineering, April 1995.
10. T. Khedro, M. Genesereth, The federation architecture for interoperable agent-based concurrent engineering systems. In *International Journal on Concurrent Engineering, Research and Applications*, Vol. 2, pages 125-131, 1994.
11. P. R. Cohen, A. Cheyer, M. Wang, S. C. Baeg, OAA: An Open Agent Architecture, AAAI Spring Symposium, 1994.
12. Y. Shoham, Agent-oriented programming, Artificial Intelligence, Vol. 60, No. 1, pages 51-92, 1993.
13. The Foundations for Intelligent Physical Agents (FIPA). http://www.fipa.org.
14. The SIGIR2006 Workshop on Personal Information Management, Seattle, WA, August 10-11, 2006. (http://pim.ischool.washington.edu/pim06)

15. D. Avrahami and S. Hudson, QnA: Augmenting an instant messaging client to balance user responsiveness and performance. *Proceeings of the ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 515-518, Jan. 2004.*
16. A. Faulring and B. Myers, Enabling rich human-agent interactions for a calendar scheduling agent. *Proceedings of the Conference on Human Factors in Computing Systems Extended Abstracts (CHI)***,** Portland, Oregon, May 2005.
17. A. Tomasic, J. Zimmerman, and I. Simmons. Linking Messages and Form Requests. *Proc. The 2006 International Conference on Intelligent User Interfaces,* Sydney, Australia, Jan. 2006.
18. N. Garera, A. Rudnicky. Briefing Assistant: Learning human summarization behavior over time. *Proc. 2005 AAAI Spring Symposium*, Jan. 2005.
19. S. Fahlman, Scone Knowledge Base. Available at: http://www.cs.cmu.edu/~sef/scone/.
20. J.P. Sousa, V. Poladian, D. Garlan, and B. Schmerl. Capitalizing on Awareness of User Tasks for Guiding Self-Adaptation. *Proc. the 1st International Workshop on Adaptive and Self-managing Enterprise Applications at CAISE'05.* Portugal, 2005.
21. Steinfeld, A., Bennett, R., Cunningham, K., Lahut, M., Quinones, P.-A., Wexler, D., Siewiorek, D., Cohen, P., Fitzgerald, J., Hansson, O., Hayes, J., Pool. M, and Drummond, M. The RADAR Test Methodology: Evaluating a Multi-Task ML System with Humans in the Loop. Carnegie Mellon University School of Computer Science Technical Report CMU-CS-06-124, CMU-HCII-06-102, May, 2006.