Proceedings of the 2006 Conference on Software Engineering and Knowledge Engineering, 5-7 July, 2006

An Architecture for Personal Cognitive Assistance

David Garlan, Bradley Schmerl
School of Computer Science, Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA, 15213
USA
+1 412 268 5057
{garlan,schmerl}@cs.cmu.edu

Abstract. 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 low-level 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 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, 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: over time specialists adapt to the needs and preferences of users.

While RADAR is the product of a large number of cooperating researchers, developed over the past three years at Carnegie Mellon, 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 outline recent empirical results of RADAR's

effectiveness in supporting a class of crisis management tasks.

2 RELATED WORK

One 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 [3]. 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 Outlook, while Calo has taken the approach of reengineering standard desktop applications to work smoothly with it's 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 [1][16][19] 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.

Another closely related area is that of Agent-oriented Architectures [7][9][11][12][13][14]. 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 [6], Zeus [5], FIPA[10]. As described later, we build on top of agent-oriented architectures (and, in particular, FIPA), specializing the general notions of agents and coordination with the specific architectural structures that characterize the RADAR architecture.

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 his behavior to rely more heavily on the PCA as he gains trust in it, the user should not be forced to do this.

Consider the following scenario. A busy user has loaded a PCA onto the 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 it should never cancel or reschedule 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:

1. **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.

2. **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
3. **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.

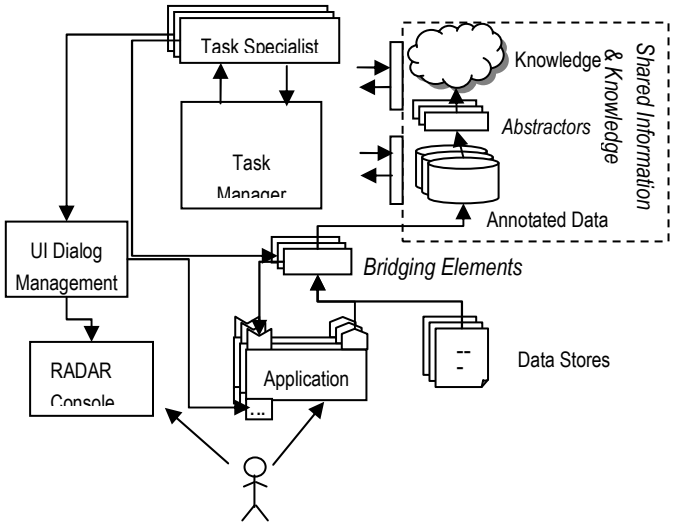

Figure 1. An Individual's Radar Architecture

4 THE RADAR ARCHITECTURE

To achieve these goals RADAR has adopted a layered runtime architecture, pictured in Figure 1, which depicts the architecture 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. Application APIs are used or written to allow RADAR to integrate with legacy application. For example, (**M**) interfaces inform RADAR of events that happen in the system (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.

On top of these RADAR adds a layer of task assistance. This can be divided into four parts:

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 each specialist to adapt to the user with respect to preferences, preferred methods of doing the task, etc. It stores this learned knowledge in the shared knowledge base.

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).

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 assis-

tance 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 representing high level entities and relationships in the user's world. For example, social nets that determine what relationships a user has to other people, are stored in a knowledge base, and 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 Figure 1. These allow RADAR specialists to monitor activities performed directly by legacy applications, and to control those applications programmatically. (We discuss the differences between categorizers and extractors below.) 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 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.

4.2 Technical Details

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."

1. Categorizers and extractors take this information and annotate it with structural information such as the positions of names (Melinda), dates (Tuesday next), and that the message is to do 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 as annotated data, with a reference to the original message.
2. Abstractors take the structured information and annotate it with knowledge. For example, it notes that Melinda is John's boss, and that John prefers meetings on Tuesday to be in the morning. The knowledge is placed in a knowledge base, which can be queried by other components.
3. The Task Management component notices that a new task to organize a meeting has been proposed, by triggers in the task database. It locates a task specialist that is responsible for managing John's calendar, and assigns the task to it.
4. 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, and by placing the new meeting on John's calendar (through the control (**C**) interface of his calendaring application in Figure 1).

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 the user, it should only do so at appropriate times. For example, in step 4 above, a calendar management specialist might want to confirm an appointment. If it immediately interrupts the user to request this, it might interrupt the user who was working on another task, causing him to lose context. For real world use, this will most likely make Fred 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 [2], based on knowledge of the user's focus and interruption policies. The UI dialog might present this information via the RADAR console, by RADAR-specific UI embellishments in legacy applications (the **UI** interface in Figure 1), or other interfaces using techniques similar to those described in [8].

A central notion of RADAR is the idea of a *task.* A task is a unit of work that the user cares about 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.[1]

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

1. *Task Dispatch and Specialist Registry*. The Task Manager acts as a directory facility for matching particular

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

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, or 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.

2. *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 details of their schedule 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, the Task Management facility is responsible for ensuring that the user's preferences are met.

3. *Inter-Radar Communication.* 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 contact their RADAR; if the person doesn't use RADAR, it chooses alternative ways to communicate (e.g., email, IM, cellphone text message).

4.3 Satisfying the Requirements

This architectural design, 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.

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.

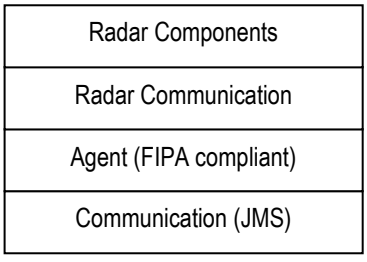

Figure 2. RADAR Layered Implementation Architecture.

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 reusable mechanisms for learning (e.g., extractors and abstractors) allow RADAR to adapt over time to a user's needs.

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.[2]

The implementation of RADAR is based on a layered use of existing technology, illustrated in Figure 2. At the lowest implementation layer are standard middleware services for distributed systems. Specifically, we used 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 layer specializes more general agent-oriented paradigms, defining specific protocols for communication between specialists and the task management services, interaction with the knowledge base, registration and invocation of the bridging elements, and the RADAR console for interacting with the user. This layer defines the rights and responsibilities for specialists, bridging elements, shared data and knowledge through a set of interface specifications.

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

[2] Although targeted for server-oriented deployment, RADAR also permits client-oriented configurations in which more of the functions run on the client side. (Indeed, our initial implementation used this configuration).

- Extractors and categorizers that understand general language terms such as places and names, but also 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 ;
  - 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;
- 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 provide the **UI**, **C**, and **M** interfaces, providing natural extension points from which to integrate Outlook with RADAR. While the interface to each legacy application will differ, our experience in another project [17] 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.

6 EVALUATION

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. The details of the evaluation are reported by others in [18]; here, we give a summary.

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 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, and 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.

Two instantiations of RADAR were used in the experiment:

1. *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 meeting. There were 31 subjects in this group.
2. *With preloaded knowledge* learned as if RADAR had been used by the conference organizer to organize the conference initially. Extractors and categorizers had been trained so that they could recognize task-related email. This group contained 47 subjects.

Test subjects engaged with the conference planning crisis scenario during two sessions of 90 minutes. In this test, learning was shown to have statistically significant positive influences on several system-wide performance metrics.

7 CONCLUSION 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. Our 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, 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.

Second, is representation and assistance with complex tasks. 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.

Third, 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. Fourth is the effective interaction between multiple RADAR spaces. At present RADAR uses standard modes of communication (such as email) to communicate between users. But it should be possible to do much better when two users both have a RADAR working in their environment.

ACKNOWLEDGEMENTS

This material is based upon work supported 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] First International Workshop on Computer Support for Human Tasks and Activties, Co-located with Pervasive 2004, Vienna, April 2004.

[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] 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.

[4] F.Bellifemine, F.Bergenti, A.Poggi, G.Rimassa, P.Turci, Middleware and Programming Support for Agent Systems. Proc. 3rd International Symposium "From Agent Theory to Agent Implementation", Austria, 2002.

[5] J.C.Collins, D.T.Ndumu, H.S.Nwana, L.C.Lee. The ZEUS agent building toolkit. *BT Technology Journal* **16**(3), 1998.

[6] P. R. Cohen, A. Cheyer, M. Wang, S. C. Baeg, OAA: An Open Agent Architecture, AAAI Spring Symposium, 1994.

[7] 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.

[8] 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.

[9] T. Finin, J. Weber, G. Wiederhold, et al., Specification of the KQML Agent-Communication Language, 1993.

[10] The Foundations for Intelligent Physical Agents (FIPA). http://www.fipa.org.

[11] 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.

[12] M. R. Genesereth, S. P. Ketchpel, Software Agents, Communications of the ACM, Vol. 37, No. 7, July 1994.

[13] 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.

[14] 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.

[15] Y. Shoham, Agent-oriented programming, Artificial Intelligence, Vol. 60, No. 1, pages 51-92, 1993.

[16] 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.

[17] 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.

[18] 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.

[19] 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.