End User Empowerment in Human Centered Pervasive Computing

End User Empowerment in Human Centered Pervasive Computing

Krzysztof Gajos, Harold Fox, and Howard Shrobe

MIT AI Lab, Cambridge, MA, USA,
{kgajos,hfox,hes}@ai.mit.edu,
http://www.ai.mit.edu/projects/iroom

Abstract. Human-centered computation is characterized by at least three features: it must be adaptive, reactive, and it must empower the user to configure and extend the behavior of the systems using her natural modes of interaction. In our previous work we have proposed systems that address the first two features. In this paper we present Alfred – a natural end user programming interface for Intelligent Environments.

1 Introduction

Our work is situated in the context of human-centered pervasive computing. Our goal is not just to make computation pervasive but also to change fundamentally the interaction between humans and computers to one in which computers adapt to human forms of discourse rather than the other way around. Human-centered computation is characterized by at least three features: first, the computational systems must be especially adaptive, responding to requests in a contextually sensitive manner. Second, they must be reactive, responding reasonably and unprompted to events in the environment. Third, end users must be empowered to modify and extend the behavior of the systems using natural interaction modes.

Ordinarily, the first two of these goals would seem to be in conflict with the third. It would seem that a complex, adaptive and reactive system would require sophisticated programming which only skilled software engineers could produce. However, the thesis of this paper, which we demonstrate in our work, is that the truth is the exact opposite. A system properly structured for self-adaptivity and reactivity to the environment provides exactly the correct vocabulary and control points for enabling end users to extend and configure the system in a "teaching by example" manner.

Within the Intelligent Room Project[6], we have previously built Rascal[5] and ReBa[7], two systems which are responsible for the adaptive and reactive components, respectively, of our software infrastructure. In this paper, we present Alfred, an end user programming interface that gives the user the ability to program the system to her particular needs and preferences.

Rascal, the core of our system, is a framework for goal-directed self-adaptivity. In brief, it works with goals, plans for achieving those goals, and the resources

needed to implement the plans. Since each goal may be satisfied by multiple plans, the system can adapt to varying circumstances by selecting a plan appropriate to the context. User preferences form a key part of the decision making about how to achieve a goal; the system should choose that method which best satisfies the user's preferences while consuming resources that have the lowest cost.

ReBa, the reactive component of our system, responds to events from the environment's perceptual systems. For example, a reaction might be triggered by someone entering a room. The task of this component is to respond to the event with a sensible behavior; for example, it might turn on the lights if someone entered the room. To maintain the overall adaptivity of the system, reactions are not handled as simple stimulus-response pairs. Instead, ReBa performs reactions by posting a new goal for Rascal to achieve; the goal is then satisfied by finding an appropriate plan, using the same techniques that are used in direct requests. For example, when a user enters his office in the morning, the system might react by posting the goal of illuminating the room. One plan for illuminating the room might turn on the lights, while another might open the drapes.

Reactions are not just straightforward mappings of events to goals, but are contextually sensitive. For example, if an office is dark because people are watching a movie, then it would be inappropriate to illuminate the room just because the door opened. In summary, ReBa handles events from perceptual signal processors and produces context-dependent reactions that posit new goals for the resource manager.

Our claim is that a system structured around these concepts provides an appropriate framework for end user empowerment. Through tools such as Alfred, it allows an end user to "program" the system by telling it the name of a new goal, demonstrating one or more plans for achieving that goal, and finally telling the system the conditions under which it would prefer one plan to another. Similarly, the user can name events that arise in the environment and tell the system what goals should be posted when those events arise. Each of these steps can be done by simple verbal commands or other natural forms of interaction. End users, in effect, record "macros" which, when situated in the infrastructure we have described, are executed adaptively and reactively.

2 More On Adaptivity In The Intelligent Room

To put our current work in context, we will briefly describe some of the technical aspects of Rascal and ReBa.

Rascal Software infrastructure for Intelligent Environments (IEs) has to be able to support interactions in a variety of spaces with very different capabilities. Rascal[5, 4], a high-level resource manager for the Intelligent Room, provides a crucial layer of abstraction by allowing applications to make high-level service requests, such as delivering a message to the user. Rascal then evaluates all available methods for satisfying the request, effectively producing a plan that

takes into account the availability of the hardware and software resources in the current environment. Additionally, Rascal can take "advice" from other agents on what kinds of resources are preferable in what context. For example, an agent detecting activity context through ReBa will discourage the use of audio devices in favor of displays when the user is talking on the phone.

This layer of abstraction allows us to design new applications without having detailed knowledge of the environments they would be running in. Currently our software is running in several offices, a conference room, a living room, and a bedroom. In those environments, the equipment ranges from a multitude of A/V devices in the conference room to a single speaker and a small display in the bedroom.

Additionally, Rascal performs arbitration among competing resource requests, adapting its resource assignments to the quantity and priority of different tasks that are running at the same time.

ReBa An Intelligent Environment is expected to react automatically to some of the events taking place within its boundaries. As mentioned in the introduction, such an environment should illuminate the room upon a person's entry. It should also take context into account when reacting to events, e.g. it should not illuminate the room upon a person's entry if the room is already occupied by people watching a movie. Finally, different kinds of spaces (such as offices or bedrooms) would have a need for very different kinds of behaviors. To address those issues, we have developed ReBa[7] to be a modular system that allows dynamic combining of context-sensitive reactive behaviors.

In ReBa, developers make individual *behavior bundles* . Each bundle defines its *activity context* and a set of responses to particular events. The activity context is composed of another set of rules that decide when a bundle should be active, i.e. the context in which its behaviors should start responding to perception events.

The core component of ReBa is its *Behavior Coordinator.* Behavior bundles register with it, and it resolves the potential conflicts and dependencies among the individual behaviors. Although all behavior bundles are written by software engineers, it is the responsibility of the owner of any individual space to choose the most appropriate combination of behaviors for his space.

As explained earlier, ReBa behaviors react to events by posting service requests to Rascal, ensuring that the reactions adapt to different environments.

3 Designing Alfred

In contrast to the above frameworks for application developers, Alfred puts the user at the center of the creation of intelligent environment functionality..

The premise of Alfred is very simple – it is essentially a multi-modal macro recorder. Upon a user's request, the system begins recording all of his actions, primarily spoken commands. When the recording is done, he assigns one or more spoken names to the recorded sequence. He can also add hardware triggers to it.

The recorded macros are simple task sequences lacking explicit conditionals. Macros can, however, call other macros, giving users the capability to create abstractions.

3.1 Interaction Decisions

The problem of designing a programming system for the end-user is extremely difficult, as explained by Nardi[9]. Techniques like visual programming, form filling, example modification, and programming by example have all been applied with moderate success in some domains, but neither one has had universal success. Despite their hype, speech, gesture, natural language, and sketching interfaces are not by themselves the solution to empowering end users' use of computation. The abstractions used in general programming are too foreign to the user's mindset for him to take the time to learn most programming tools. What works is a tool whose high-level structures and formalisms match the formalisms present in the user's own mind. Programs like spreadsheets and CAD tools have been very effective end-user tools despite their complexity and inhuman syntax. This is because these programs use primitives and constructions that are natural from the education and experience of the accountants and designers that use them.

While interacting with Alfred, the user performs actions in a task sequence exactly as she would when she performs the tasks individually. A sequence of tasks forming a procedure is quite familiar to most users from recipes and other instructions. A metaphor for the interaction is that of a user instructing a butler (hence the name of the system). The input modality we chose is a spoken dialogue with the environment as this is the primary modality in the Intelligent Room[1].

3.2 Software Architecture

In order to explain the technical underpinning of Alfred, we will first briefly describe the relevant parts of the software infrastructure behind the Intelligent Room.

Metaglue[2] is a multi-agent system that uses two communication mechanisms: direct method calls and a publish-subscribe mechanism.

Speech recognition[3] is done centrally in each environment, but the processing of recognized utterances is distributed among all speech enabled agents. When the speech recognition engine recognizes a spoken utterance, it passes the utterance to the appropriate agent via a direct method call. It also posts a speech recognition event containing the details of the spoken utterance as well as the name of the destination agent.

As explained earlier, user commands cause high-level service requests to be sent to Rascal. Those requests are evaluated dynamically based on which services are available in a given environment and which are currently unused. Even such low level agents as the light manager use a level of indirection to discover and adapt to the illumination capabilities of whatever physical space they are operating in.

Given this infrastructure, in the recording phase, Alfred can observe a user's deliberate actions (such as speech commands, device controlling actions, etc) and record them. The information contained in the event notifications is sufficient to later on "recreate" the events.

Our speech infrastructure allows instantenous updates of vocabulary so new named task sequences can be accessed as soon as they are recorded. When a task is triggered – either by a direct command or by a hardware trigger – Alfred contacts appropriate agents as if the previously recorded commands and events just took place.

4 Using Alfred

Recording A New Macro The following interaction illustrates how Alfred can be taught a new task sequence.

User: I want to record a new macro.

Computer: Beginning to record a macro. Say "stop recording" when you are done.

User: Turn on the main lights. Open the drapes. Turn on my desk lamp. Say "good morning." Stop recording.

Computer: What phrase would you like to associate with this macro?

User: "Good morning, computer."

Computer: Any other phrase?

User: No, I am done.

Computer: Macro added!

From this moment on, any time the user says "Good morning, Computer," the above sequence of actions will be repeated.

Adding A Hardware Trigger Task sequences can also be associated with hardware events. In particular, we have added a number of "free buttons" to our conference room that can be programmed for any task. Tasks may also be associated with existing hardware interfaces, such as the light switch:

User: When I press this button [user presses one of the free buttons] run the "Good morning, computer" sequence.

Computer: Please press the button again for confirmation.

[User presses the button again]

Computer: Done!

From this moment on, whenever somebody presses the button, the "Good morning, computer" macro will be automatically executed.

Invoking The Tasks A recorded task sequence can be invoked in three different ways: through a spoken command, through a hardware trigger (if defined), and through a graphical user interface (if present). Our conference room is equipped with a touch panel where interfaces for a number of agents are displayed. Alfred also displays a list of task sequences it has been taught. Users can activate any of

the sequences by pressing the right button on the display. Our handheld devices can also obtain Alfred's GUI upon entering a space.

Occasionally, Alfred encounters errors while executing task sequences. Generally, errors occur when there is no resource available that can provide one of the recorded tasks or when the component performing an individual task reports an error. In such cases, Alfred tells the user which of the tasks failed and asks if it should attempt to run the remaining tasks or if it should abort the sequence.

5 Evaluation

Alfred has been developed very recently, and we have not yet conducted formal user studies to evaluate it. We have, however, deployed the system in a conference room and two offices. In both offices and in the conference room, the users have recorded a number of task sequences. Most of them are for reconfiguring the spaces for different kinds of activities: meetings, demonstrations, presentations or movie watching in the conference room, and working, meeting or presentation in the offices. In the individual offices, the users prefer to invoke tasks using speech. In the conference room, the primary modality is the graphical interface displayed on a touch panel by the entrance to the room. As a rough guide, we have also applied the nine heuristics proposed by Nielsen[10] to avoid major flaws in our design.

6 Future Direction

Alfred allows users to easily extend and customize the capabilities of their environments. However, several future directions suggest themselves immediately to make end-user programs more expressive. Specifically, we want to add support for conditional statements. Beyond this, we would like to enable users to create multiple solutions to their requests and to create preferences for one solution over another in different contexts.

Extending Alfred to use conditionals is straightforward. The same events that we use to trigger service requests in ReBa can be used to make user-defined service requests act differently under different system state. For example, as part of the "Good morning" task sequence, the user can say: "If the temperature is greater than eighty-five, turn on the fans. Otherwise, if the temperature is less than sixty-five, turn on the heater."

Another way we can make user-defined service requests more dynamic is to allow multiple plans to satisfy a request, with user preferences dictating the execution of one over the other. For example, I can set up two plans to light up the room: "turn on the lights" and "open the drapes". To specify a preference for the latter over the former, I can say: "To light up the room, if it is daytime, I prefer plan one to plan two. Otherwise, I prefer plan two to plan one." A method for efficiently computing utility functions from sets of such simple preferences has been recently proposed by McGeachie[8]. In our example, if the user is in a windowless conference room, and she says "light up the room", only the "turn

on the lights" request is available. If she is in a windowed office with lights and drapes, there are two plans available, and the appropriate utility function for each of them gets computed from the recorded preferences. During the day, when she says "light up the room," her agent will open the drapes. By providing multiple solutions to any given service request, the user can instruct her agent to adapt to different rooms and environments with different resource capabilities.

7 Conclusions

In this paper we have described a system that is highly adaptive and reactive while supporting end user empowerment at the same time. Indeed, we have argued and demonstrated that the structures and conceptual framework defined to support adaptivity and reactivity provide the appropriate control points to enable the end user to program the system by direct instruction.

When the end user programs the system, she defines a goal and a set of plans, composed of sub-goals, that could be used to satisfy the goal. Her preferences dictate which plans are weighted higher than others. In this way, the user can create new functions and new reactions that can migrate easily from one space to another.

References

1. Michael Coen. Design principles for intelligent environments. In *Fifteenth National Conference on Artificial Intelligence (AAAI98)*, Madison, WI, 1998.
2. Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The Metaglue system. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
3. Michael Coen, Luke Weisman, Kavita Thomas, and Marion Groh. A context sensitive natural language modality for the Intelligent Room. In *Proceedings of MANSE'99*, Dublin, Ireland, 1999.
4. Krzysztof Gajos. A knowledge-based resource management system for the Intelligent Room. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.
5. Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of CEEMAS 2001*. Springer, 2001. To appear.
6. Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *Proceedings of ABA 2002*, July 2002. To Appear.
7. Ajay Kulkarni. A reactive behavioral system for the Intelligent Room. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2002.
8. Michael McGeachie and Jon Doyle. Efficient utility functions for ceteris paribus preferences. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, August 2002. To Appear.
9. Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, 1993.
10. Jacob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of ACM CHI'90*, Seattle, WA, 1990.