

---

# Plan-Driven Ubiquitous Computing

---

Gary Look  
Stephen Peters

GARYL@AI.MIT.EDU  
SLP@AI.MIT.EDU

MIT Computer Science and Artificial Intelligence Laboratory, 200 Technology Square, Cambridge MA, 02139 USA

## 1. Introduction

Recent work in ubiquitous computing (“ubiquitous computing”) research has made it easier to create user-centered applications that adapt to the different resources available in various computing environments. However, these systems, such as Aura (Garlan et al., 2002), have a limited representation of the larger scope in which these resources are used by an individual. Consequently, many ubiquitous computing applications operate in a myopic mode that lacks the ability to proactively assist the user in his task. To provide ubiquitous computing applications with a representation of what a user is actually doing and how the application can assist him, we describe a plan-driven execution model for ubiquitous computing and describe a system implementation of this model.

All of our work is in situations within a variety of intelligent environments (IEs) ranging from kiosks to personal offices to meeting rooms. However, all these IEs are driven by a common software base. This makes it particularly appealing to abstract away from the details of the specific devices, focusing instead on the common goals that one might bring into any of these facilities, the abstract plans that can achieve such goals, and the characteristics of the resources needed to populate each such plan. By applying techniques from the AI literature to the area of ubiquitous computing, we have built a system that allows for this to happen.

## 2. Scenario

To illustrate the value of plan-directed execution, we describe a scenario our system handles.

Steve has just received a reminder that one of his undergraduate students has scheduled time today to discuss research progress. As the student enters, Steve issues a command to his office for it to properly configure itself for this session. Since the discussion is likely to consist of brainstorming new ideas, the room turns on microphones that can record the discussion, and also enables the device that captures drawings and text written on the large whiteboard next to his desk. The room makes sure the environment’s lighting is suitably bright and then begins to record the meeting as it begins. When the meeting is over, the envi-

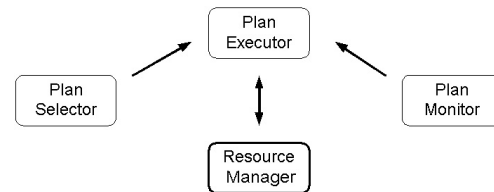


Figure 1. System architecture of our plan-driven paradigm.

ronment turns off the microphone and whiteboard devices, and stores the captured information for later review.

Steve has other kinds of informal meetings in his office, and each of these require different resources. Our system selects an appropriate plan for each situation and controls the IE accordingly.

## 3. System Overview

Our overall paradigm is that one makes a request for a goal to be achieved together with a set of preferences saying what properties are important to the requester (e.g., speed is more important than privacy). These preferences are then used to guide the selection of a plan capable of achieving the goal and to find an allocation of resources needed by the plan.

Figure 1 presents an architectural overview of the system that implements this paradigm. The system consists of four main components: a plan selector that chooses which plan best achieves a user’s goal, a resource manager that assigns resources to these plans, a plan executor that carries out the plan, and a plan monitor that records plan progress. As the plan is carried out, the plan executor may consult the resource manager since conditions may change, thus requiring a new resource assignment to the plan. In this abstract, we focus on the resource management and plan execution and monitoring components of the system. The plan selector uses a utility function based on user preferences (McGeachie & Doyle, 2002) to choose which plan the executor should carry out.

## 4. Plans As a Representation of User Tasks

A plan is a decomposition of a task into a set of partially-ordered steps. Each step has a set of pre-conditions that must be met before the step can be performed and a set of post-conditions that hold after. Each step in a plan may require a set of resources subject to certain constraints. Since there is often more than one set of resources capable of meeting the plan's needs, the choice of which to use is late-bound to best accommodate the actual operating conditions at execution time. Lastly, the plan as a whole and the individual steps in the plan have a set of annotations. An annotation can be any information that is relevant to carrying out a plan, such as which plan steps require special attention because they are more likely to fail and recommended orderings of steps.

In our system, we use Planlet (Look, 2003), a middleware layer written in Java, to represent plans and user progress through plans. In addition to capturing the basic plan features described above, Planlet also allows plans to be defined hierarchically and for plans to be modified as they are being carried out. These two features allow plans to be reused by other plans and also allow for steps to be "late-bound" to a plan, if it is beneficial to defer the choice of a specific plan of action until execution time.

## 5. Resource Management

The plan executor need not specify the precise devices that it will be using during the plan execution. Instead, we rely on a resource management capability that allows plans to specify their needs in terms of more abstract resource descriptions (for example, the plan could request a "projector display"). If the requester desires, it can also provide contextual information (such as "the room is currently in a 'meeting' mode"), which the resource manager can use to assist in arbitration.

Our resource management module utilizes a semantic network knowledge base (Peters & Shrobe, 2003). This "Semantic Network Resource Module" (SNRM) allows the selection of favored resources, dependent on highly contextual situations, inside the environment's knowledge representation. For example, users can describe situations like "In his office, Steve usually uses the first projector, except when he's having a meeting, when he uses the second" by setting up two different preference resources. SNRM then chooses which projector to use based on which preference resource matches the current situation most accurately.

## 6. Plan Execution and Monitoring

Once a specific plan has been chosen to execute, and resources have been assigned to the plan, the IE then carries

out the plan. Before a step is executed, the plan executor interprets the step and any annotations the step may have and then dispatches the necessary information to the software components that can complete the step. In our implementation, steps that the IE can carry out consist of a generic resource description, the name of a method offered by the service and a list of arguments to pass to that method.

If a user is the one who is performing a task, the plan monitor must be able to determine when he is finished with the task. There are a number of approaches to making this determination, such as using perceptual techniques to observe and infer that the user is done with a step. At the moment, we adopt a simpler approach to plan monitoring and listen for explicit user cues to indicate that he has completed a task, such as a button press or a vocal statement like, "I'm done."

## 7. Contributions

We have described how adapting AI planning and rational decision-making techniques to ubicomp applications makes these applications more adaptive to a user's intended plan and preferences. Our overall paradigm is goal and plan driven; the user provides a high-level request for a goal to be achieved together with a set of preferences saying what properties are important to the requester. These preferences are then used to guide the selection of a plan that best meets the user's needs and to find and allocate the resources needed by that plan. This plan is then enacted by a plan executive that dispatches sub-tasks as they become enabled and a plan monitor records their progress. The combination of explicit plan representations and late-binding, dynamic decision making allows the system to respond to variations in the execution environment in an adaptive manner.

## References

- Garlan, D., Siewiorek, D., Smailagic, A., & Steenkiste, P. (2002). Project Aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing*, 1, 22–31.
- Look, G. (2003). Plan-Based Proactive Computing. Masters of Science Thesis, Massachusetts Institute of Technology.
- McGeachie, M., & Doyle, J. (2002). Efficient utility functions for ceteris paribus preferences. *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (pp. 279–284).
- Peters, S., & Shrobe, H. (2003). Using semantic networks for knowledge representation in an intelligent environment. *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*.