Planning, Scheduling, and Plan Execution for Autonomous Robot Office Couriers

Michael Beetz and Maren Bennewitz
University of Bonn, Dept. of Computer Science III,
Roemerstr. 164, D-53117 Bonn, Germany,
email: beetz, bennewit@cs.uni-bonn.de

Abstract

This paper investigates issues in the integration of planning, scheduling, and plan execution in the context of control systems for autonomous robot office couriers. Such control systems have to meet challenging requirements. They must avoid late deliveries and achieve long-term efficiency instead of restricting themselves to the optimization of problem solving episodes. Because robots acting in human working environments are often forced to schedule their activities without having complete information their control systems must reliably and efficiently execute scheduled activity in changing and partly unknown situations. In particular, the control systems must be able to exploit opportunities, flexibly avoid problems, and integrate command revisions for currently active tasks.

We propose to implement such control systems for scheduled activity by employing concurrent reactive plans that, while performing their actions, reschedule the course of action whenever necessary. The plans have a modular and transparent representation which permits easy transformations by transformation rules. It is these rules that implement the controller's scheduling and schedule repair methods.

Introduction

To carry out their jobs reliably and efficiently many autonomous mobile service robots acting in human working environments have to view their jobs as everyday activity. We consider a particular instance of everyday activity: performing office courier service. Scheduling everyday activity differs in important aspects from many other scheduling tasks such as job shob scheduling (FS84), space shuttle scheduling (DSB94), or transportation scheduling in the following aspects:

- ***Amortized cost and utility of plans.*** The goal of scheduling everyday activity is the optimization of long-term efficiency rather than problem-solving episodes. Therefore, a competent office courier distributes for instance empty envelopes according to an expected consumption profile while performing its delivery jobs. Distributing the envelopes beforehand decreases the chances that the robot must search for empty envelopes before delivering a letter and thereby miss a deadline. Such preparation actions make the performance of individual jobs slower but they can be expected to improve the overall performance significantly.
- ***Flexibility and Robustness.*** Schedules are to be generated based on partial information about the environment and the tasks. For instance, incomplete task specifications like "pick up the letter from Wolfram and deliver it," lack proper descriptions of envelope as well as the destination of the letter. Acting appropriately based on partial information requires the robot courier to watch out for opportunities and exploit them as well as detect and avoid problems while executing scheduled activity.
- ***Experience.*** Information acquired through extended experience is exploited to compute more appropriate schedules. For instance, the time it takes for different people to load or unload at different places.

It is important that the scheduler of an autonomous robot office courier is able to interleave delivery jobs, reschedule when problems are detected, and exploit opportunities. The scheduler also has to be able to predict whether exploiting an opportunity that has just been detected might cause failures in other activity threads such as missing deadlines (BG98). What seems less important is the computation of schedules that guarantee minimal path length because loading and unloading takes significant amount of time.

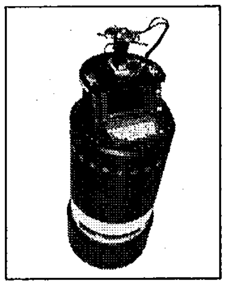


Fig. 1: The mobile robot RHINO

Our research on scheduling everyday activity is carried out in the context of FAXBOT, a structured reactive controller (SRC) (Bee98) that is designed for robust and efficient execution of delivery plans on the autonomous mobile robot RHINO (see Fig. 1), an RWI B21 robot. RHINO is equipped with three PCs connected to the university-wide computer network via a tether-less radio link. Its sensor system contains 24 ultra-sonic proximity sensors, two laser range finders, and a CCD stereo color camera system.

FAXBOT operates in a part of an office building containing a large hallway, several offices, a library, and a classroom (see Figure 2). FAXBOT uses a symbolically annotated 3D world model of its environment that contains floor plan information for walls, doorways, and rooms and static pieces of furniture. The world model provides all the information required by the RHINO navigation system to exhibit fast and collision free navigation and accurate robot localization (TBB+98). It also contains detailed geometric models of the furniture items that can be used for their recognition. Finally, the world model stores symbolic information that is used to interpret natural language commands and as domain knowledge for mission planning.

FAXBOT operates over extended periods of time and carries out a schedule of multiple jobs, which can be changed at any time. Jobs are issued via electronic mail using strongly restricted natural language.

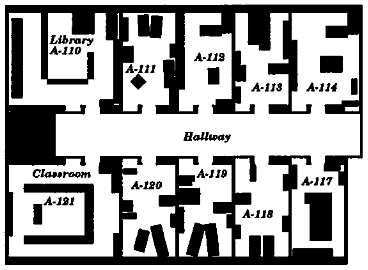


Fig. 2: Environment of the office courier

To accomplish its commands FAXBOT uses a library of routine plans. The routine plan for deliveries specifies that RHINO is to navigate to the pickup place, wait until letter is loaded, navigate to the destination of the delivery, and wait for the letter to be unloaded.

The main contributions of this paper are that we show (1) the installation of modular and transparent schedules in complex concurrent and reactive robot control programs; (2) explain the design of schedules and controllers that allow for the opportunistic and robust execution of scheduled activity; and (3) describe novel plan transformation techniques for scheduling and rescheduling everyday activity.

The remainder of this paper is organized as follows. The section 2 presents an example in which FAXBOT schedules its delivery jobs based on partial information and reschedules its activities to avoid task failures and exploit opportunities. The section 3 gives a glimpse of the plan representation used by FAXBOT and presents the control structures used for the implementation and revision of activity schedules. Section 4 presents technical details of FAXBOT's activity scheduler and shows how scheduling methods are implemented as declarative plan transformation rules. Section 5 describes how the FAXBOT controller accomplishes the behavior of the robot that is described in Section 2.

An Extended Example

Consider the following experiment that is carried out by RHINO using FAXBOT's scheduling capabilities. RHINO receives two commands: "put the red letter on the meeting table in room A-111 on the desk in room A-120" and "deliver the green book from the librarian's desk in room A-110 to the desk in room A-114."

Whenever the jobs change FAXBOT computes an appropriate schedule very fast. FAXBOT's initial schedule is to pick up the red letter first, then pick up the green book, then deliver the red letter, and the green book afterwards. If the second job involved also a red letter to be delivered, then the scheduler would first carry out the first job completely and then carry out the second job. This is done to avoid carrying two red letters at the same time and thereby yield possible confusions.

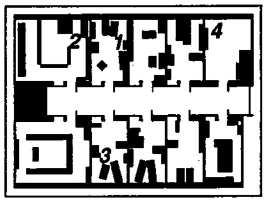


Fig. 3. Default schedule.

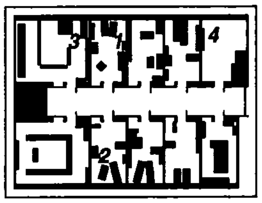


Fig. 4. Schedule that avoids carrying two red letters.

Proposing a schedule also implies making assumptions about whether and when doors are open or closed. Thus when adopting the schedule FAXBOT assumes that the doors of the rooms A-110, A-111, A-114, and A-120 are open. To perform the necessary adaptations flexibly, FAXBOT monitors the assumptions underlying its schedule while performing its deliveries: whenever it passes a door it estimates the opening angle of that door and revises the schedule if necessary.

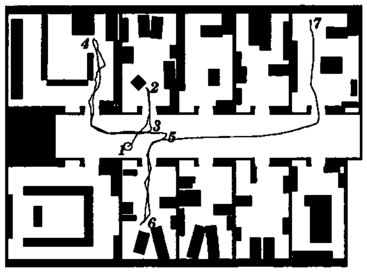


Fig. 5: Complete trajectory for the two deliveries

Figure 5 pictures RHINO's trajectory during the accomplishment of the two delivery jobs. Initially, all doors in the environment are open. FAXBOT starts with the delivery of the red letter and heads to the meeting table in A-111 where the letter is loaded (step 2). At this moment the door of A-120 is closed. Thus, when FAXBOT enters the hallway to deliver the red letter at Michael's desk, it estimates the opening angle of the door of room A-120. At this moment FAXBOT detects that the door has been closed and that it cannot complete the delivery (step 3). A failure is signalled.

Since FAXBOT does not know when room A-120 will be open again, it revises the schedule such that it delivers the green book first and accomplishes the failed delivery as an opportunity. Thus FAXBOT navigates to the librarian's desk in A-110 to pick up the green book to room A-114 (step 4). At this moment room A-120 is opened again. As FAXBOT heads towards A-114 to deliver the green book it passes room A-120 (step 5). At this point the door estimation process signals an opportunity: A-120 is open! Therefore, FAXBOT interrupts its current delivery to complete the delivery of the red letter. After the delivery of the red letter is completed (step 6), FAXBOT continues the delivery of the green book (step 7). The behavior generated by FAXBOT if all doors stay open is shown in Fig. 6 and the one if A-120 is closed but not opened again in Fig. 7.

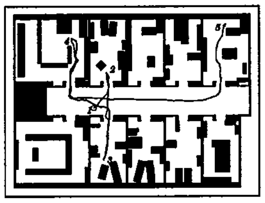


Fig. 6. Trajectory if A-120 stays open.

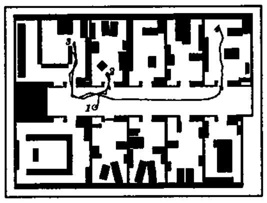


Fig. 7. Trajectory if A-120 is closed again.

The behavior exhibited by FAXBOT demonstrates the following capabilities of FAXBOT's scheduler and its integration into the overall controller: interleaving delivery tasks, revising schedules while they are executed, and exploiting opportunities.

The FAXBOT Controller

FAXBOT's delivery routines are implemented in RPL (Reactive Plan Language) (McD91). RPL provides conditionals, loops, program variables, processes, and subroutines. RPL also places high-level constructs (interrupts, monitors) to synchronize parallel physical actions and make plans reactive and robust by incorporating sensing and monitoring actions, and reactions triggered by observed events at the programmer's disposal. The RPL constructs used to specify scheduled activity are the PLAN-, WITH-POLICY-, WHENEVER-, and WAIT-FOR-statements; but see (McD91) for a complete description.

The PLAN-statement has the form (PLAN *STEPS CONSTRAINTS*). *STEPs* have the form (:TAG *NAME SUBPLAN*) and tag *SUBPLAN* with the name *NAME*. *constraints* have the form (:ORDER S_1 S_2) where S_is are name tags of the plan steps. *Steps* are executed in parallel except when they are constrained otherwise. The :ORDER constraints make sure that a subsequent step is started only if all steps computing the inputs have been completed. The TOP-LEVEL command indicates declares its subtasks as user commands and causes the generation of failure and success reports upon their termination.

WITH-POLICY *P B*, another control structure, means "execute the primary activity *B* such that the execution satisfies the policy *P*." Policies are concurrent processes that run while the primary activity is active and interrupt the primary if necessary.

Events that require RHINO to perform actions such as "passing a door" are handled through fluents, program variables that signal changes of their values and thereby enable control threads to react to asynchronous events. The RPL statement WHENEVER *F B* is an endless loop that executes *B* whenever the fluent *F* gets the value "true." WAIT-FOR *F*, another control abstraction, blocks a thread of control until the fluent *F* becomes true.

Primary Activities and Policies

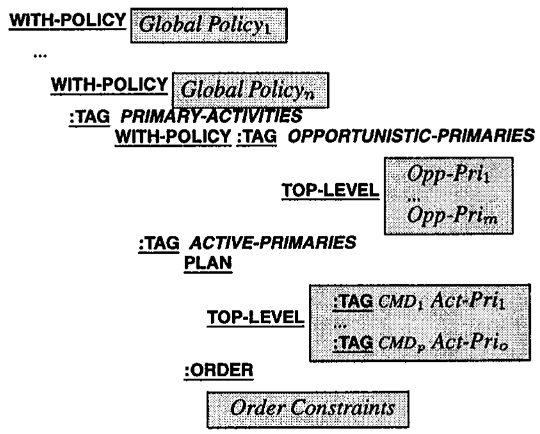
The FAXBOT controller carries out two kinds of subplans: *primary activities*, actions taken to accomplish the robot's mission and *policies*, which monitor and maintain the conditions necessary for the successful and efficient execution of the primary activities.

Primary activities include for example the navigation to places where objects are to be picked up and delivered. A policy might, for instance, monitor the doors the robot is passing to detect whether they are open or closed. Another policy might monitor how well the robot knows its position and invoke active localization whenever necessary. Primary activities must handle interrupts and, due to the possible side-effects of policies, these activities have to make suitable preparations for their successful continuation after reactivation.

Policies are best viewed as constraints on the execution of primary activities. Constraints such as "whenever you pass a door estimate the opening angle of the door using its laser range finders" and opportunities such as "complete the delivery to room A-120 as soon as you learn the office is open," which are both necessary for carrying out the jobs opportunistically, are specified using the RPL construct WITH-POLICY. Events like "passing a door" are detected by fluents that trigger actions like estimating the door angle. When a delivery gets interrupted because FAXBOT has detected that the door to A-120 is open, that opportunity has the side effect of moving the robot into the office A-120. The interrupted delivery plan has therefore to be replanned before it can be continued.

The Structure of the FAXBOT Controller

The FAXBOT controller is structured in a modular and transparent way such that automatic plan transformation techniques can retrieve parts of the plan easily and modify the plan without making it opaque for subsequent plan revision processes.



```
WITH-POLICY  Global Policy_1
 ...
  WITH-POLICY  Global Policy_n
   :TAG PRIMARY-ACTIVITIES
       WITH-POLICY :TAG OPPORTUNISTIC-PRIMARIES
                                    Opp-Pri_1
                         TOP-LEVEL  ...
                                    Opp-Pri_m
         :TAG ACTIVE-PRIMARIES
             PLAN
                              :TAG CMD_1 Act-Pri_1
                   TOP-LEVEL  ...
                              :TAG CMD_p Act-Pri_o
             :ORDER
                       Order Constraints
```

The overall plan of the FAXBOT controller consists of the plan body tagged PRIMARY-ACTIVITIES that contains the plans for accomplishing the user commands and the surrounding policies that specify the constraints on the execution of the primary activities. The primary activities are separated into the opportunistic primaries and the active primaries. The active primaries are the ones that the robot is able to accomplish without help. The order in which the subplans of the active primaries are executed is given by the *order constraints* that specify a (partial) order on the navigation tasks contained in the active primary tasks. The opportunistic primaries are the ones that robot cannot accomplish autonomously. To complete them it has to wait for enabling conditions. For example, because FAXBOT has no action for opening doors it might have to wait for doors to open in order to complete its deliveries. The open door might be an opportunity to complete a user command.

The FAXBOT Scheduler

The scheduling method takes constraints, resources, and deadlines into account and tries to minimize the number of interruptions for people. The method carries out a sequence of steps: (1) the scheduler extracts all navigation tasks from the structured reactive plan; (2) it computes an efficient total order on the navigation tasks that (if possible) meets all given deadlines avoids overloading the robot; (3) it installs the schedule into the structured reactive plan; and (4) installs a monitoring process that monitors all the assumptions made by the scheduler and triggers a rescheduling process whenever an assumption is detected to be violated.

The extraction step is simple because all navigation tasks are required to have the form (ACHIEVE (LOC RHINO *loc*)) (BM92). In the extraction step every navigation task is tagged with a unique name that can be later used to specify the order on the navigation tasks. A more difficult issue in the extraction step is guessing the destinations of navigation tasks that are not completely specified. (McD92) describes a technique for guessing destinations of navigation tasks based on Monte Carlo simulations of the plan.

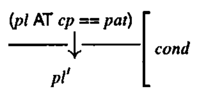
The algorithm for ordering the navigation tasks is also simple. Essentially, it sorts the destination north of the hallway in ascending and south of the hallway in descending order (the environment is pictured in Figure 2). After this initial sort the scheduler iteratively resolves problems such as missed deadlines or confusions caused by carrying objects that look identical.

```
SCHEDULE-DELIVERIES($delivs$)
for each $Loc$ in $Delivs$
  do if $North\text{-}of?(Loc, Hallway)$
        then $North \leftarrow North \cup \{Loc\}$
        else $South \leftarrow South \cup \{Loc\}$
$North \leftarrow Sort(North, \geq)$
$South \leftarrow Sort(South, \leq)$
for each $Loc$ in $North$
  do if $East\text{-}of?(Loc, Robot)$
        then $East \leftarrow East \cup \{Loc\}$
        else $West \leftarrow West \cup \{Loc\}$
$Schedule \leftarrow Append(West, South, East)$
while $\neg Conflict\text{-}free(Schedule)$
  do $Schedule \leftarrow Remove\text{-}a\text{-}Conflict(Schedule)$
return $Schedule$
```

Because of the hierarchical structure of office environments this simple heuristic algorithm for computing schedules works surprisingly well. The optimal schedule asks the robot to visit every office with the order of the offices being clockwise (or anti-clockwise) starting at the office closest to the robot. The Voronoi diagrams of offices have often almost tree structure (with few cycles) so that the desks can be easily reached from the door and the center of the room.

There are various cases for which this algorithm produces suboptimal schedules. For instance, you can place n destinations on a circle (equally spaced). In this setting, the algorithm asks the robot to go back and forth on the circle and take a route that is longer than necessary. The algorithm can be extended easily so that it computes with a high probability nearly optimal schedules fast. The question is whether having nearly optimal schedules causes the robot to accomplish its tasks more efficiently. Often tighter schedules fail more often. In our opinion it is more promising to learn scheduling heuristics from experience. Information relevant for scheduling that can be learned from experience include the expected duration of loading and unloading, when people are usually in their offices, when doors are open or closed, and so on.

The scheduling routine is implemented as a plan transformation rule that can be applied to FAXBOT's overall plan while the plan is executed (BM97; BM94). We will diagram plan-transformation rules



```
1.) ?PRIM-ACTIVITIES                                          UNSCHEDULED-PLAN-BUG                          (a)
        AT PRIMARY-ACTIVITIES                                   (?ROOT-TASK ?ACTIVE-PRIMARIES)
2.) ?ACTIVE-PRIMARIES                                         ∧NAVIGATION-TASKS                             (b)
        AT ACTIVE-PRIMARIES                                     (?ACTIVE-PRIMARIES ?NAV-TASKS)
        == (PLAN ?CMDS ?CONSTRAINTS)                          ∧SCHEDULE ?NAV-TASKS(?SCHED-CONSTRAINTS)      (c)
------------------------------↓------------------------------ ∧SCHEDULING-ASSUMPTIONS                       (d)
1.) WITH-POLICY WHENEVER (NOT (AND !?SCHED-ASSMTS))             (?NAV-TASKS ?SCHED-ASSMTS))
                    SIGNAL(UNSCHEDULED-PLAN-BUG
                           (?ROOT-TASK ?ACTIVE-PRIMARIES))
      ?PRIM-ACTIVITIES
2.) PLAN ?CMDS !?CONSTRAINTS !?SCHED-CONSTRAINTS
```

Figure 8: Plan revision rule for the the installation of task schedules.

where *cond* is the applicability condition, (*pl* AT *cp* == *pat*) the input plan schema and *pl'* the output plan schema of the rule. The applicability condition is a conjunction of literals. The input plan schema consists of a pattern variable to which the subplan with code path *cp* is bound and an optional pattern *pat* that is matched against the subplan *pl*. The rule is applicable if the application condition holds and the plan *pl* with code path *cp* matches the pattern *pat*. The resulting plan fragment *pl'* replaces the input fragment in the revised plan.

Figure 8 shows the formalization of the transformation rule that schedules office delivery tasks. The transformation rule revises the primary activities by adding another global policy that generates an unscheduled plan bug whenever an assumption underlying the current schedule is detected as violated. The rule also revises the active primaries by adding the ordering contraints of the schedule to the constraints of the primary activities. The scheduling rule is applicable under a set of conditions specifying that (a) There is a bug of the category "unscheduled plan;" (b) The navigation tasks contained in the active primaries are ?NAV-TASKS; (c) ?SCHED-CONSTRAINTS are ordering constraints on ?NAV-TASKS such that any order which satisfies ?SCHED-CONSTRAINTS will accomplish the active primary tasks fast and avoid deadline violations and overloading problems; (d) ?NAV-TASKS can be accomplished if ?SCHED-ASSTS are satisfied. The rule is applied whenever the set of user commands changes.

Robust and Efficient Schedules

To accomplish robust and efficient execution of schedules we (1) explicitly record and monitor scheduling assumptions; (2) install opportunistic delivery tasks; and (3) reschedule the primary activities while they are executed whenever a broken scheduling assumption is detected.

Currently, we only consider one kind of scheduling assumptions and opportunities: the state of doors. To monitor these assumptions, the FAXBOT controller employs a global policy: whenever the robot passes a door it estimates the opening angle of this door. The opening angles of each door are stored in an fluent that has two dependent fluents: one signalling that the door is closed and one that the door is open.

The scheduling assumptions of the active primaries are that all rooms that are destinations of navigation tasks are open. Scheduling assumptions are handled by a policy of the following form:

```
WITH-POLICY SEQ WAIT-FOR(CLOSED(ROOM_1) ∨...
                                ∨CLOSED(ROOM_n))
                       SIGNAL(CLOSED-DOOR-BUG ...)
      Plan Body
```

Thus each violated scheduling assumption triggers the application of the scheduling transformation rule shown in Figure 9. In order to apply scheduling transformations to the primary activities while the primary activities are executed, the primary activities have to be restartable (BM96). Restartability means that the robot controller can repeatedly start executing a plan, interrupt the execution, and start it anew, and the resulting behavior of the robot is very similar to the behavior of executing the plan only once. Restartability facilitates the smooth integration of plan revisions into ongoing activities: a partially executed restartable plan can be revised by terminating the plan and starting the new plan that contains the revisions.

In practice, however, restartable plans work differently. Consider, for example, a plan for the delivery of an object. To determine which parts of the plan can be skipped, it is sufficient to know the object's location and some aspects of the robot's state (like its location and what it carries). Because the FAXBOT controller updates the object descriptions whenever it perceives and "manipulates" the object, FAXBOT can determine the state of execution based on the description of the object to be delivered.

To act efficiently, FAXBOT has to exploit opportunities. A necessary precondition of exploiting opportunities is that FAXBOT is able to specify what opportunities are. One kind of opportunity that FAXBOT can exploit is the opportunistic completion of delivery tasks. Suppose a user command cannot be completed because the room where the object is to be delivered is closed. In this case, the execution of the active primary plan for accomplishing the delivery fails and the plan is installed as an opportunistic primary.

The revision of the FAXBOT controller that is triggered

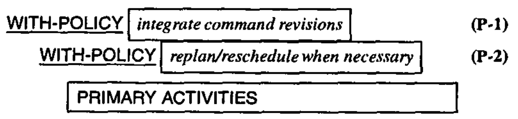
```
1.) ?ACT-PRIMARIES
        AT ACTIVE-PRIMARIES
2.) ?OPP-PRIMARIES                             CLOSED-DOOR-BUG
        AT OPPORTUNISTIC-PRIMARIES               (?CMD-FAILURE ?DOOR-FCT ?ROOM
                                                  (?TLC) ?FAILED-TASK)
                                               ∧TLC-NAME(?TLC ?TLC-NAME)
                                               ∧ROOT-TASK(?ROOT ?FAILED-TASK)
1.) ?REM-ACT-PRIMARIES                         ∧DELETE-ACTIVE-PRIMARY-TLC-PLAN
2.) TOP-LEVEL                                     (?ROOT ?TLC-NAME ?REMAINING-TLC-PLANS)
      :TAG ?TLC-NAME                           ∧TAGGED-SUBTASK(?ROOT ?TLC-NAME ?TLC-TASK)
           SEQ WAIT-FOR(OPEN(?ROOM))           ∧RPL-EXP(?TLC-TASK ?TLC-PLAN)
               ?TLC-PLAN
      !?OPP-PRIMARIES
```

Figure 9: Plan revision rule for delivery tasks that cannot be completed because of closed doors.

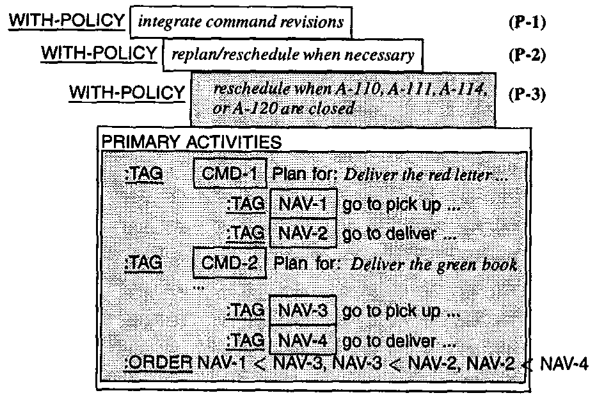
by closed doors is accomplished by the plan transformation rule listed in Figure 9. The plan revision rule is triggered by a "closed door" bug that causes a user command to fail. The rule deletes the failed plan for the user command from the active primary activities and adds the plan to the opportunistic primary activities.

FAXBOT's Scheduling Methods at Work

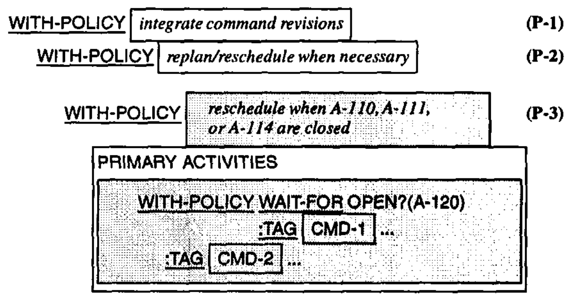
This section describes how FAXBOT accomplishes the courier jobs described earlier. In the beginning, FAXBOT carries out no primary activities. Its outermost policy ensures that new commands are received and processed.


WITH-POLICY integrate command revisions (P-1)
WITH-POLICY replan/reschedule when necessary (P-2)
PRIMARY ACTIVITIES

Upon receiving the two commands the policy **P-1** puts plans for the commands into the active primary activities of the SRC. The insertion of the commands triggers the scheduler of the policy **P-2** that orders the navigation tasks in the primary activities. The scheduling policy also adds an additional policy **P-3** that monitors the assumptions underlying the schedule, that is that the rooms A-110, A-111, A-114, and A-120 are open.


WITH-POLICY integrate command revisions (P-1)
WITH-POLICY replan/reschedule when necessary (P-2)
WITH-POLICY reschedule when A-110, A-111, A-114, or A-120 are closed (P-3)
PRIMARY ACTIVITIES
:TAG CMD-1 Plan for: Deliver the red letter ...
:TAG NAV-1 go to pick up ...
:TAG NAV-2 go to deliver ...
:TAG CMD-2 Plan for: Deliver the green book ...
:TAG NAV-3 go to pick up ...
:TAG NAV-4 go to deliver ...
:ORDER NAV-1 < NAV-3, NAV-3 < NAV-2, NAV-2 < NAV-4

After FAXBOT has picked up the red letter from the meeting table and left room A-111, it notices that room A-120 has been closed in the meantime. Because FAXBOT cannot complete the delivery of the red letter the corresponding command fails. This failure triggers the replanning policy **P-3**. Because FAXBOT cannot foresee when room A-120 will be open again it transforms the completion of the delivery into an opportunity. Thus as soon as FAXBOT notices room A-120 to be open it interrupts its current mission, completes the delivery of the red letter, and continues with the remaining missions after the red letter has been successfully delivered.


WITH-POLICY integrate command revisions (P-1)
WITH-POLICY replan/reschedule when necessary (P-2)
WITH-POLICY reschedule when A-110, A-111, or A-114 are closed (P-3)
PRIMARY ACTIVITIES
WITH-POLICY WAIT-FOR OPEN?(A-120)
:TAG CMD-1 ...
:TAG CMD-2 ...

Discussion

Our research on scheduling the activities of an autonomous robot office courier are still in an early stage. So far we have only performed some simple experiments to validate that the FAXBOT controller and its scheduler work; that is that it can reliably monitor scheduling assumptions and schedule delivery jobs during their execution (see our example in the second section). In the future we plan to compare the behavior generated by the FAXBOT controller with alternative controllers that apply different (re)scheduling strategies. We also need to examine more carefully the literature on robust scheduling, in particular (ZF94).

In this paper we have mainly focussed on the application of scheduling techniques to plans that control an autonomous mobile service robot. As such the contributions of this paper lie mainly in the representation of complex concurrent and reactive plans that facilitate scheduling operations, the specification of plan revision methods in the form of plan transformation rules, and the application of these scheduling methods while the robot carries out the scheduled activity.

FAXBOT accomplishes its jobs successfully because its subplans are made interruptable and restartable using high-level control structures that specify synchronized concurrent reactive behavior. FAXBOT achieves adaptivity through plan revision and scheduling processes, implemented as policies, that detect opportunities, contingent situations, and invalid assumptions. Plan revision techniques are able to perform the required adaptations because of the modular and transparent specification of concurrent and reactive behavior. In particular the distinction of policies and primary activities increases the modularity significantly. Policies enable FAXBOT to specify opportunistic behavior and to achieve reliable operation while making simplifying assumptions.

There are many open issues that we would like to investigate more carefully in the near future. These issues include the development of more sophisticated scheduling methods (ZF94), the application of learning techniques to acquire useful information that can be exploited by heuristic scheduling methods (HC92b; HV98a; HV98b; ZDD$^+$92), and a thorough experimental investigation on the effects of different scheduling techniques on the behavior exhibited by autonomous service robots (HC92a).

References

M. Beetz. Structured reactive controllers for service robots in human working environments. In G. Kraetzschmar and G. Palm, editors, *Hybrid Information Processing in Adaptive Autonomous Vehicles*. Springer, 1998. to appear.

M. Beetz and H. Grosskreutz. Causal models of mobile service robot behavior. In R. Simmons, M. Veloso, and S. Smith, editors, *to appear in Fourth International Conference on AI Planning Systems*, Morgan Kaufmann, 1998.

M. Beetz and D. McDermott. Declarative goals in reactive plans. In J. Hendler, editor, *First International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1992.

M. Beetz and D. McDermott. Improving robot plans during their execution. In Kris Hammond, editor, *Second International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1994.

M. Beetz and D. McDermott. Local planning of ongoing behavior. In Brian Drabble, editor, *Third International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1996.

M. Beetz and D. McDermott. Expressing transformations of structured reactive plans. In *Recent Advances in AI Planning. Proceedings of the 1997 European Conference on Planning*, pages 64–76. Springer Publishers, 1997.

M. Drummond, K. Swanson, and J. Bresina. Scheduling and execution for automatic telescopes. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, pages 341–369. Morgan Kaufmann Publishers, 1994.

M. S. Fox and S. Smith. Isis - a knowledge-based system for factory scheduling. *Expert systems*, 1(1):25–49, 1984.

D. Hart and P. Cohen. Predicting and explaining success and task duration in the phoenix planner. In J. Hendler, editor, *AIPS-92: Proc. of the First International Conference on Artificial Intelligence Planning Systems*, pages 106–115. Kaufmann, San Mateo, CA, 1992.

A. Howe and P. Cohen. Isolating dependencies on failure by analyzing execution traces. In J. Hendler, editor, *AIPS-92: Proc. of the First International Conference on Artificial Intelligence Planning Systems*, pages 277–278. Kaufmann, San Mateo, CA, 1992.

K. Haigh and M. Veloso. Learning situation-dependent costs: Improving planning from probabilistic robot execution. In *To appear in Autonomous Agents 98*, 1998.

K. Haigh and M. Veloso. Planning, execution and learning in a robotic agent. In *to appear in Fourth International Conference on AI Planning Systems*, 1998.

D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998. to appear.

M. Zweben, E. Davis, B. Daun, E. Drascher, M. Deale, and M. Eskey. Learning to improve constraint-based scheduling. *Artificial Intelligence*, 58:271–296, 1992.

M. Zweben and M. S. Fox. *Intelligent Scheduling*. Morgan Kaufmann, 1994.