Generating, Executing and Revising Schedules for Autonomous Robot Office Couriers

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

Abstract

Scheduling the tasks of an autonomous robot office courier and carrying out the scheduled tasks reliably and efficiently pose challenging problems for autonomous robot control. 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: they should accomplish long-term efficiency rather than optimize problem-solving episodes. They should also exploit opportunities and avoid problems flexibly because often robots are forced to generate schedules based on partial information.

We propose to implement the controller for scheduled activity by employing concurrent reactive plans that reschedule the course of action whenever necessary and while performing their actions. The plans are represented modularly and transparently to allow for easy transformation. Scheduling and schedule repair methods are implemented as plan transformation rules.

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. Thus, scheduling everyday activity differs from many other scheduling tasks such as job scheduling (FS84), space shuttle scheduling (DSB94), or transportation scheduling in the following aspects:

- ***Long-term efficiency.*** The goal of scheduling everyday activity is the optimization of long-term efficiency rather than problem-solving episodes. In certain situations, for instance, a competent office courier distributes empty envelopes according to an expected consumption profile while performing its delivery jobs. Distributing the envelopes beforehand decreases the chances that the robot has to pickup empty envelopes before delivering a letter and therefore misses a deadline. Such preparation actions make the performance of individual jobs slower but they can be expected to improve the overall performance significantly.
- ***Robustness and Flexibility.*** 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's desk and deliver it," lack proper descriptions of the envelope as well as the destination of the letter. In such a case, the robot has to postpone the execution until more information is available. Acting flexibily requires the robot courier to watch out for opportunities and exploit them as well as to detect and avoid problems while executing scheduled activity.
- ***Experience.*** Information acquired through extended experience is exploited to compute more appropriate schedules. For instance, knowledge about the time needed by individual users to upload or unload items can be exploited to estimate the required overall time more accurately.

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.

We propose to implement the controller for scheduled activity by employing concurrent reactive plans that reschedule the course of action whenever necessary and while performing their actions. The plans are represented modularly and transparently to allow for easy transformation. Scheduling and schedule repair methods are implemented as plan transformation rules.

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.

Fig. 1: RHINO

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 opportunistic and robust execution of scheduled activity; and (3) describe novel plan transformation techniques for scheduling and rescheduling everyday activity.

FAXBOT's Scheduling Methods at Work

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 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. WITH-POLICY *P B* 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. Primary activities must handle interrupts and, due to the possible side-effects of policies, have to make suitable preparations for their successful continuation after reactivation.

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.

Consider the following experiment that is carried out on RHINO using FAXBOT's scheduling capabilities (details on the scheduling and plan transformation methods can be found in (BB98)). RHINO receives two commands: "put the red letter on the meeting table in room A-111 onto 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" (see Fig. 2).

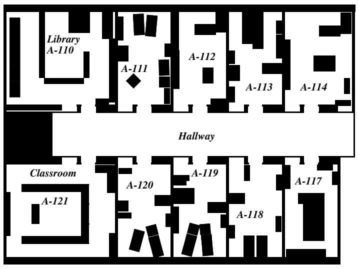

Fig. 2: Environment of the office courier

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.

WITH-POLICY *integrate command revisions* **(P-1)**
WITH-POLICY *replan/reschedule when necessary* **(P-2)**
PRIMARY ACTIVITIES
:TAG CMD-1 Plan for: *Deliver the red letter*
:TAG CMD-2 Plan for: *Deliver the green book*

The insertion of the commands triggers the scheduler of the policy **P-2** that orders the navigation tasks in the primary activities.

Figure 3 shows the formalization of the transformation rule that schedules office delivery tasks. The transformation rule revises the primary activities by adding another policy that hat generates a bug whenever an assumption underlying the current schedule is detected as violated. The rule also revises the active primaries by

```
1.) ?PLANNED-POLICIES                                   UNSCHEDULED-PLAN-BUG                      (a)
       AT PLANNED-POLICIES                                  (?ROOT-TASK ?ACTIVE-PRIMARIES)
       == (PAR !?CURR-PLANNED-POLICIES)                 ∧ ACTIVE-NAVIGATION-TASKS                 (b)
2.) ?MAIN-PLAN                                                  (?ROOT-TASK ?NAV-TASKS)
       AT MAIN-PLAN                                     ∧ OPEN-DOOR-ASSUMPTIONS                   (c)
       == (PLAN ?STEPS !?CURR-CONSTRAINTS)                  (?NAV-TASKS ?OPEN-ASSMTS))
----------------------------↓----------------------     ∧ DOOR-ASSUMPTIONS-MONITOR                (d)
1.) PAR ?MONITOR !?CURR-PLANNED-POLICIES                    (?OPEN-ASSMTS ?MONITOR))
2.) PLAN ?STEPS !?CURR-CONSTRAINTS !?SCHED-CONSTRAINTS  ∧ PROPOSED-SCHEDULE                       (e)
                                                            (?NAV-TASKS ?SCHED-CONSTRAINTS)
```

Figure 3: Plan revision rule for the the installation of task schedules.

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) ?NAV-TASKS can be accomplished if ?OPEN-ASSMPTS are satisfied; (d) ?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. The rule is applied whenever the set of user commands changes.

The algorithm for ordering the navigation tasks sorts the destinations on one side of the hallway (see Fig. 2) in ascending and the others in descending order. After this initial sort, the scheduler iteratively resolves problems such as missed deadlines or confusions caused by carrying objects that look identical.

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).

In our example, the application of the revision rule yields:

```
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,                    (P-3)
               or A-120 are closed]
   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 the door of 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**.

Each violated scheduling assumption triggers the application of the closed-door-transformation rule shown in Figure 4.

The revision rule of the FAXBOT controller that is triggered by closed doors (see Figure 4) causes the corresponding 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. This is done by adding another policy that watches out for the door of A-120 to be opened again.

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

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.

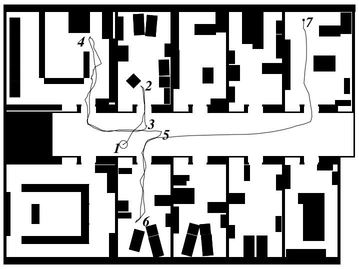

Fig. 5: Complete trajectory for the two deliveries

Figure 5 pictures RHINO's trajectory during the accomplishment of the two delivery jobs. 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

```
1.) ?ACT-PRIMARIES                              CLOSED-DOOR-BUG
        AT ACTIVE-PRIMARIES                        (?CMD-FAILURE ?DOOR-FCT ?ROOM
2.) ?OPP-PRIMARIES                                 (?TLC) ?FAILED-TASK)
        AT OPPORTUNISTIC-PRIMARIES              ∧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 4: Plan revision rule for delivery tasks that cannot be completed because of closed doors.

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). 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.

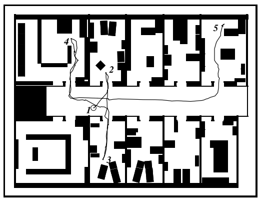

Fig. 6. Trajectory if A-120 stays open.

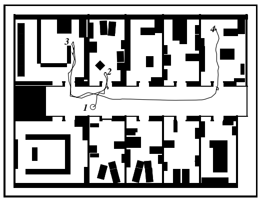

Fig. 7. Trajectory if A-120 is closed again.

Discussion

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 by means 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.

Our research is still at 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.

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 and M. Bennewitz. Planning, scheduling, and plan execution for autonomous robot office couriers. In *submitted for publication*, 1998.

M. Beetz. Structured reactive controllers. In *submitted for publication*, 1998.

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. 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. 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 AIPS-98*, 1998.

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

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.