Plan Management for Robotic Agents

Michael Beetz

Autonomous robots that perform complex jobs in changing environments must be capable of managing their plans as the environmental conditions or their tasks change. This raises the problem of deciding whether, when, where, and how to revise the plans as the robots' beliefs change. This article investigates an approach to execution time plan management in which the plans themselves specify the plan adaptation processes. In this approach the robot makes strategical (farsighted) adaptations *while* it executes a plan using tactical (immediate) decisions and overwrites tactical adaptations after strategical decisions have been reached (if necessary). We present experiments in which the plan adaptation technique is used for the control of two autonomous mobile robots. In one of them it controlled the course of action of a museums tourguide robot that has operated for thirteen days and performed about 3200 plan adaptations reliably.

1 Introduction

Autonomous service robots such as office couriers [SGH97] or museum tourguides [BCF98] have become challenging test-beds for developing and testing computational models of competent agency. One of the key problems in controlling autonomous robotic agents that are to solve complex jobs in changing environments is the management of their "intentions", that is the plans they are committed to execute [PH99]. While autonomous robots execute their plans their beliefs about the world change continually and therefore the robots must permanently decide whether to keep executing the plans they have committed to, revise them, or generate and commit to new ones.

Consider, a robot office courier that has committed to a course of action. In the midst of carrying out a planned delivery tour the robot learns, while passing an office, that a door that it previously assumed to be closed is in fact open. What should the robot do? Should it simply ignore the open door and continue its intended course of action? Or, should it consider the open door as an opportunity to achieve its jobs in a better way? A robot, passing offices at walking speed, must decide quickly. Taking an open door as an opportunity might cause subtle side effects (eg, delays or the objects that the robot will be carrying) in the future part of the robot's intended course of action. Typically, the robot hasn't planned for such contingencies. Therefore, the robot should also reason carefully through the consequences of such decisions.

Different solutions have been proposed for this kind of decision problems. The first category of solutions tries to eliminate the need for making sophisticated execution time decisions altogether. The idea is to precompute all possible decisions into a universal plan [Sch87]. This kind of approach has recently gained a lot of attention. A number of approaches transform planning problems into (partially observable) Markov decision problems ((PO)MDP) and compute the optimal actions for all possible states [BDH98]. The universal plan and (PO)MDP approach, however, is only feasible if the state spaces are small or at least well structured.

The approaches that make execution time decisions that are not precomputed can be divided into those that only make situation-specific decisions without foresight and those that deliberate as much as necessary before they decide. Situation-specific decisions without foresight can be made very quickly [Fir89]. The problem, however, is that locally optimal decisions might be suboptimal in the global context. The second solution is to generate a complete plan for the new situation from scratch [BCF98]. This "stop and replan" strategy is the best solution as long as planning is very fast. Often, however, determining plans with high expected utility under uncertainty will take seconds or even minutes – no matter whether or not it will result in a better course of action.

Thus, a number of more pragmatic solutions between these poles have been proposed. One of those is to dynamically plan only for a small subset of the robot's goals. For example, Simmons *et al.* [SGH97] only consider goals with the highest priority and tasks that can be accomplished on the way. This approach searches for local optima but fails to recognize global consequences of decisions. Layered architectures [BFG97] run planning and execution at different levels of abstraction and different time scales where a sequencing layer synchronizes between both levels. In these approaches the plans provide only guidelines for execution. Disadvantages of these approaches are that planning processes use models that are too abstract for predicting all consequences of the decisions they make. In addition, planning processes cannot exploit the control structures provided by the lower levels for specifying more flexible and reliable behavior. Yet other approaches make no committment on how to make such decisions [IGR92]. Finally, approaches for real-time search and resource-bounded reasoning [Zil96] trade off the resources needed to compute better plans against the performance gains that can be expected from them.

This article presents an approach to runtime plan management which is based on the following ideas:

1. Plan management is performed by ***self-adapting plans*** that contain so-called ***plan adaptors***. The robot monitors the beliefs about the assumptions underlying its plan. Plan adaptors are triggered by specific ***belief changes***. Upon being triggered the adaptors decide whether adaptations are necessary and if so perform them.
2. Runtime plan management is carried out as a two stage process that first makes a fast ***tactical adaptation*** that decides on how to continue in this very moment, and immediately starts a ***strategic adaptation process*** that may later revise the tactical adaptation. The strategical decision reasons through the possible consequences of plan adaptations and performs revisions based on foresight.

3. Plan adaptation processes are specified explicitly, modularily, and transparently and are implemented using ***declarative plan transformation rules***.
4. The plans that the plan adaptation processes reason about and revise are ***adaptable***. They are designed for allowing the plan adaptation processes fast and transparent access to the relevant plan pieces. Adaptable plans are also capable of continuing their execution after revisions.

The most salient advantages of our computational model of runtime plan management are:

- Plan adaptation processes are specified as subplans. As a consequence, ***all*** decision making is represented explicitly within the plan and not hidden in a separate sequencing system, which enables the robot to add, delete, and revise in the same way as other plan steps. Furthermore, the interaction between plan adaptors and the plans is explicitly specified in the plan.
- Implementing plan management as a combination of tactical and strategical adaptations also has advantages. The robot can exploit that many delayed decisions, if they require revisions at all, can be incorporated into a plan easily because they only affect subplans that have not yet been started. The robot therefore does not have to stop to replan and is always under the control of a plan.

2 The Plan Management Framework

We develop our framework for runtime plan management within the computational model of ***structured reactive controllers*** (**SRC**s) [Bee99]. SRC s are collections of concurrent reactive control routines that adapt themselves to changing circumstances during their execution by means of planning. SRC s employ and reason about plans that specify and synchronize ***concurrent percept-driven*** behavior. The plans are represented in a transparent and modular form so that automatic planning techniques can reason about them and revise them. SRC s are implemented in RPL (Reactive Plan Language) [McD91]. The reactive control structures provided by SRC s are used to specify how to react to changes of its beliefs.

The remainder of this Section describes and discusses the components of our plan management framework, which is a rational reconstruction of the use of plan revision methods in SRC s [Bee99] and is also implemented using transformational planning of reactive behavior [McD92].

Plan Adaptors. The plan management methods of a plan adaptor are divided into tactical adaptations that are performed instantaneously and strategic adaptations that might take a considerable amount of computation time to arrive at a decision. Thus, a plan adaptor is defined by

```
define plan adaptor           name   (args)
    :triggering-belief-change bc
    :tactical-adaption        t-adpt*
    :strategical-adaption     s-adpt*
```

where *bc* is the triggering belief change, *t-adpt** a possibly empty set of tactical and *s-adpt** a possibly empty set of strategical adaptation methods.

Because plan adaptations might have global consequences that cannot be foreseen easily, tactical decisions are sometimes shortsighted. Therefore, plan adaptation is implemented as a two step process: first a tactical plan adaptation is performed which buys time and then a strategical adaptation is made that might overwrite the tactical decision. Of course, there are cases in which stopping the execution of the current plan and waiting for a new plan is the best execution strategy. This is just a special case of a plan adaptor, in which the tactical adaptation inserts an additional plan step into the plan that blocks the further execution until the strategical adaptation methods have reached a decision on how to continue.

Many technicalities of how to implement the planning processes that are performed as part of the strategical plan adaptations are described in [BBG99].

Adaptable Plans. Our plan management framework uses plan adaptation methods that make strong assumptions about plans to simplify the computational problems. As a consequence, the methods can apply reliable and fast algorithms for the construction and installment of subplans, the diagnosis of plan failures, and for editing subplans during their execution. Making assumptions about plans is attractive because plan adaptation methods construct and revise the plans and can thereby enforce that the assumptions hold. To enforce that a self-adapting plan satisfies a property *Q* throughout its existence it is sufficient that it satisfies *Q* in the beginning and that its plan adaptors preserve *Q* [Bee99]. A robot office courier that schedules its activities in order to shorten the paths the robot has to follow, for example, might enforce that the locations at which actions have to be performed are represented explicitly and the actions to be performed at the same locations collected modularily in subplans. These properties facilitate runtime plan scheduling drastically.

Self-adapting Plans are specified using the construct

```
with plan adaptor   Adpt
        Pl
```

which means carry out plan *Pl* but whenever the triggering belief of the plan adaptation process *Adpt* changes *Adpt* is executed. For example, the following piece of plan

```
with plan adaptor   handle-unexpected-open-door ({A-110, A-120})
   delivery plan
```

tells the robot to carry out *delivery plan* but if it learns that office A-110 or A-120 has become open it runs the plan adaptation procedure *handle-unexpected-open-doors*.

Because self-adapting plans are plans themselves, other plan adaptors can reason about, delete, insert, or revise them. For example, a global scheduler can add ordering constraints on the execution of the individual delivery steps but also plan adaptors for rescheduling that are triggered if the robot learns that assumptions underlying the schedule are violated.

Plan Adaptation Rules. The plan management capabilities are realized through plan revision rules of the following form.

```
if      cond
then    transform $ips_1$ at $cp_1$
        into $ops_1$
        ...
        transform $ips_n$ at $cp_n$
        into $ops_n$
```

where *cond* is the applicability condition, ips_i at cp_i the input plan schema and ops_i 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_i is bound. cp_i specifies the path in the syntax tree where the subplan to be revised can be found. The rule is applicable if the application condition holds. The resulting plan fragment ops_i replaces ips_i.

3 Plan Management in RHINO

This section describes the application of the plan management framework discussed in the last section to the control of an autonomous robotic office courier called RHINO. We focus on plan adaptations for generating and revising schedules.

RHINO's Adaptable Plans. To facilitate online rescheduling RHINO's plans are modularized with respect to the locations where subplans are to be executed using the at location *plan schema*. The at location ⟨*x,y*⟩*p* plan schema that is stored in RHINO's plan library is a good example of making concurrent, percept-driven plans modular and transparent. It specifies that plan *p* is to be performed at location ⟨*x,y*⟩. Here is a simplified version of the plan schema for at location.

```
named subplan  N_i
  do at location  ⟨x,y⟩   p   by
    with valve  Wheels
      do with local vars  DONE? ← FALSE
        do loop
          try in parallel
            wait forTask-Interrupted?(N_i )
            sequentially
              do  NAVIGATE-TO ⟨x,y⟩
                  p
                  DONE? ← TRUE
        until DONE? = TRUE
```

The plan schema accomplishes the performance of plan *p* at location ⟨*x,y*⟩ by navigating to the location ⟨*x,y*⟩, performing subplan *p* and signalling that *p* has been completed (the inner sequence). The with valve statement blocks the semaphore *Wheels* that any process changing the location of the robot must own. The loop makes the execution of *p* at ⟨*x,y*⟩ robust against interruptions from higher priority processes. Finally, the named subplan statement gives the subplan a symbolic name that can be used for addressing the subplan for scheduling purposes and in plan revisions.

Using the at location plan schema, a plan for delivering an object *o* from location *p* to location *d* can be roughly specified as a plan that carries out *pickup(o)* at location *p* and *putdown(o)* at location *d* with the additional constraint that *pickup(o)* is to be carried out before *putdown(o)*.

We call a plan ***location transparent*** if and only if every subplan *p* that is to be performed at a particular location *l* has a symbolic name and is reduced from a plan of the form at location ⟨*x,y*⟩ *p*. In addition, ordering conditions on at location subplans are either specified in the form of RPL statements or explicit :order clauses (see below).

For location transparent plans a scheduler can traverse the plan recursively and collect all names of the at location subplans, the locations where they are to be performed, and the ordering constraints on these subplans. The scheduler then determines an extension on these ordering constraints that maximizes the expected utility of the overall plan and installs these ordering constraints in the plan.

Another important precondition for the successful realization of runtime plan management is that the controller allows for smooth integration of revisions into ongoing scheduled activities. Again, we will achieve this feature through a careful design of the plans in the plan library. Plans that satisfy this feature are called ***restartable***, which simply means that each subplan keeps a record of its execution state and if started from anew skips the parts of the plan that do not have to be executed any more.

We can make a plan for a single delivery restartable by equipping the plan *p* with a variable storing the execution state of *p* that is used as a guard to determine whether a subplan is to be executed. The variable has three possible values: *to-be-acquired* denoting that the object must still be acquired; *loaded* denoting that the object is loaded; and *delivered* denoting that the delivery is completed.

Thus, the whole delivery plan can be skipped if the execution state is „delivered." Otherwise a sequence of two steps is executed. The pickup step is only executed if the object is not loaded already. The second step is executed if the object is still loaded. To handle interruptions by a concurrent plan the execution state must be updated before the interrupting plan gets the control. Beetz and McDermott[BM96] describe an approach dealing with these kinds of problems.

```
if  ¬  EXECUTION-STATE (p, delivered)
  then sequentially
    do if  EXECUTION-STATE (p, to-be-acquired)
      then  AT-LOCATION     L     PICK-UP(o)
    if  EXECUTION-STATE (p, loaded)
      then  AT-LOCATION     D     PUT-DOWN(o)
```

A plan for a delivery tour has the form plan *steps orderings* where *step*s are instances of the delivery plan schema. *constraint*s have the form :order s_1 s_2 where s_is are name tags of subplans (eg, the name tags of the at location subplans :order s_1 s_2 means that s_1 must be completed before s_1 is started. *steps* are executed in parallel except when they are constrained otherwise.

We call a tour plan for a set of delivery commands restartable if for any sequence of activations and deactivations of the tour plan ending with a plan activation of the last plan step holds that (1) every pickup and putdown operation gets executed; (2) the order in which the subplans are executed satisfy the ordering constraints; (3) every pickup and putdown operation gets executed at most once.

Observation: RHINO's default delivery plans are ***location transparent*** and ***restartable.***

RHINO's Plan Adaptors. Concurrently with the scheduled activity the high-level controller runs plan adaptors that monitor and handle belief changes that might warrant plan adaptations. RHINO's controller reacts to the following belief changes: (1) situations in which doors that are expected to be open are detected to be closed and vice versa; (2) asynchronously arriving command additions, revisions, and deletions; (3) situations in which the colors of letters to be delivered are different from their expected colors. We plan to extend the scope of situations to unexpected delays and unexpected fast progress.

```
PLAN ADAPTATION RULE UNSCHEDULED-PLAN-STRATEGY
IF BELIEF-CHANGE(OPEN-DOOR(?ROOM))
   ^ ACTIVE-PRIMARIES(?ACT-PRIMS)
   ^ AT-LOC-PLANS(?ACT-PRIMS ?AT-LOC-TASKS)
   ^ GOOD-SCHEDULE(?AT-LOC-TASKS ?SCHED-CONSTRS)
   ^ SCHEDULING-ASSMPTS (?NAV-TASKS ?SCHED-ASSMPTS))
THEN TRANSFORM ?PRIM-ACTS AT PRIMARY-ACTIVITIES
   INTO WITH PLAN ADAPTOR
         WHENEVER: ¬ ∧(?SCHED-ASSMTS)
            SIGNAL(UNSCHEDULED-PLAN-BUG(?ACT-PRIMS))
         ?PRIM-ACTS
   TRANSFORM ?ACT-PRIMS AT ACTIVE-PRIMARIES
   INTO PLAN ?CMDS (?CONSTRS ∪ ?SCHED-CONSTRS)
```

Figure 1: Strategic plan adaptation rule.

Fig. 1 shows an adaptation rule that strategically reschedules office delivery tasks whenever the robot learns about an open door. The rule revises the plan 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 plan. The scheduling rule is applicable under a set of conditions specifying that (a) There is a belief change considering a door that was assumed to be closed but is indeed open; (b) The *at location* tasks contained in the active primaries are ?NAV-TASKS; (c) ?SCHED-CONSTRS are ordering constraints on ?NAV-TASKS such that any order which satisfies ?SCHED-CONSTRS 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 heuristic inference needed to compute instantiations for ?SCHED-CONSTRS such that GOOD-SCHEDULE holds is more sophisticated. In fact, making this inference requires a planning process. The planning technique that is performed is called probabilistic prediction-based schedule debugging (PPSD) [BBG99] .

PPSD iteratively revises a given schedule until it has computed a satisfying schedule. The hill-climbing search that decides whether a new candidate schedule is better than a previous one is based on sampling a small number of possible execution sce-narios for each schedule and therefore has a risk of being false. Even for sampling few execution scenarios, making one iteration requires several seconds. Thus, PPSDis too slow to be applied as a plan adaptation tactics.

```
PLAN ADAPTATION RULE CLOSED-DOOR-TACTICS
IF BELIEF-CHANGE(OPEN-DOOR(?ROOM))
   ^ USUALLY-CLOSED(?ROOM) ^TLC-NAME(?TLC ?TLC-NAME)
   ^ DELETE-ACT-PRIMARY-TLC-PLAN
      (?TLC-NAME ?REMAINING-TLC-PLANS)
   ^ NAMED-SUBTASK(?TLC-NAME ?TLC-TASK)
   ^ RPL-EXP(?TLC-TASK ?TLC-PLAN)
THEN TRANSFORM ?ACT-PRIMS AT ACTIVE-PRIMARIES
      INTO ?REM-ACT-PRIMARIES
      TRANSFORM ?OPP-PRIMS AT Opportunistic-Primaries
      INTO TOP-LEVEL
         :TAG ?TLC-NAME
            SEQ WAIT-FOR(OPEN(?ROOM))
               ?TLC-PLAN
         ?OPP-PRIMARIES
```

Figure 2: Revision rule CLOSED-DOOR-TACTICS

Fig. 2 shows a tactical adaptation rule. The rule is triggered by "closed door" belief changes and applies to doors that are often closed. The rule deletes the failed plan for the user command from the scheduled activities and adds it to the opportunities. Whenever the triggering condition of an opportunity is satisfied, it interrupts the active activity, completes its execution, and passes the control back to the interrupted activity. Note that all conditions of the adaptation tactics can be tested quickly. An important property of the revision rules is that they keep the plans locational transparent and restartable. Both rules neither change at location subplans nor introduce new plans that are to be performed at particular locations. So the resulting plans are still locational transparent. Because the scheduling algorithm produces a schedule in which all completed plan steps are before the plan steps that remain to be carried out, the plans resulting from the rule Unscheduled-Plan-Strategy are restartable if the original plan was restartable. Finally, the Closed-Door-Tactics keeps the plan restartable under the condition that the triggering condition becomes true while the plan gets executed. In general, all plan adaptation rules that RHINO applies pre-serve locational transparency and restartability.

PPSD keeps the plan restartable under the condition that the triggering condition becomes true while the plan gets executed.

In general, all plan adaptation rules that RHINO applies preserve locational transparency and restartability.

4 Experiments

This Section describes several experiments that evaluate the reliability and flexibility of runtime plan adaption and possible performance gains that it can achieve.

The Minerva Experiment. We have run an SRC as a high-level controller for MINERVA, an interactive museums tourguide robot that has operated for a period of thirteen days in the Smithsonian's National Museum of American History [TBB99]. In this period, it has been in service for more than ninetyfour hours, completed 620 tours. RPL controlled MINERVA's course of action in a feedback loop that was carried out at three Hertz. MINERVA used plan adaptation methods for the installment of new commands, the deletion of completed plans, and tour scheduling. MINERVA performed about 3200 plan adaptations. The MINERVA experiment demonstrates that SRC s can (1) reliably control an autonomous robot over extended periods of time and (2) reliably revise plans during their execution. The plan management framework described here is a rational reconstruction of MINERVA's plan revision mechanisms.

Performing Office Delivery Tasks with Runtime Plan Adaptation. Consider the following experiment that is carried out by the mobile robot RHINO. In the beginning, RHINO carries out no primary activities. It runs a process that receives new commands and the plan adaptor **P-1** that integrates plans for the newly received tasks into its overall plan.

RHINO receives two commands shown in 3(a). In the first one RHINO is asked to get a book from Wolfram's desk in A-111 to Jan's desk in A-113. The second one requires RHINO to deliver a letter from "nobody's" desk in A-117 to the library (room A-110). Whenever the jobs change RHINO the plan adaptor computes an appropriate schedule and installs it. Upon receiving the two commands the adaptor **P-1**puts plans for the commands into the plan. The insertion of the commands triggers a second adaptor **P-2** that orders the AT-LOCATION tasks. The

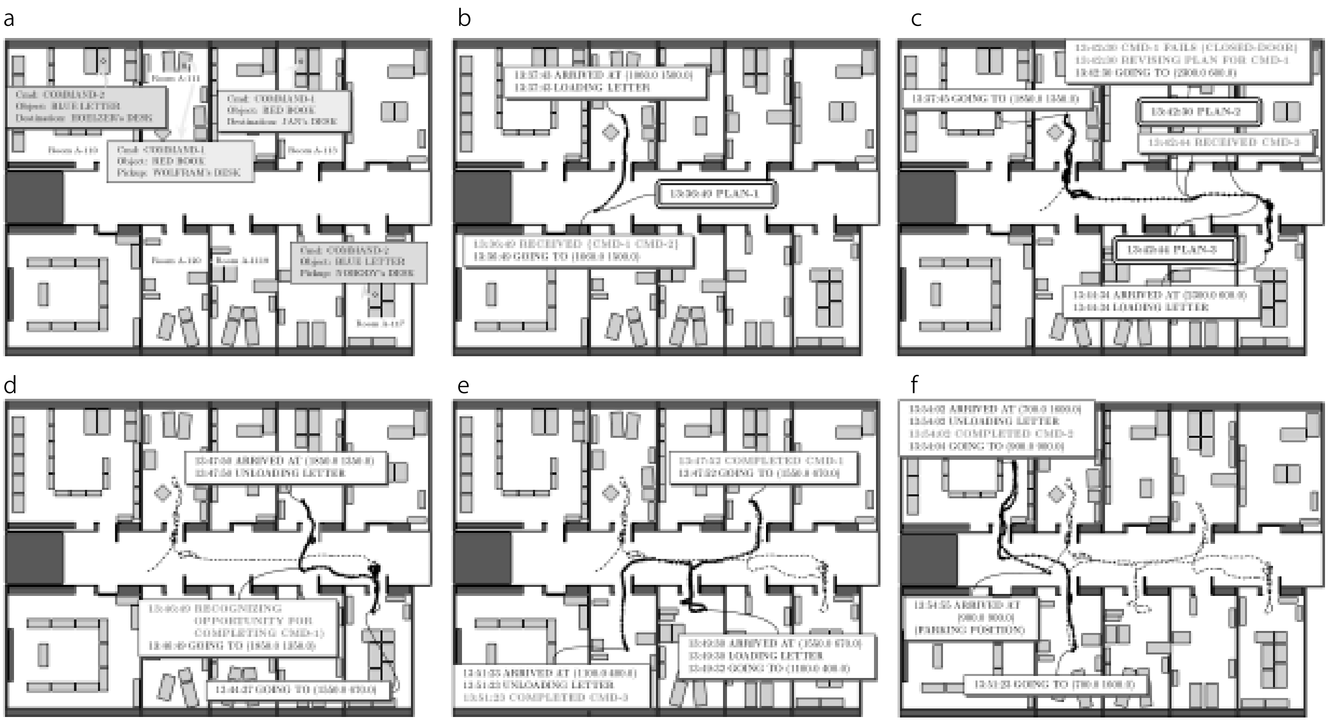

Figure 3: Execution trace for the second experiment

scheduling plan adaptor also adds an additional adaptor **P-3** that monitors the beliefs underlying the schedule, that is that the rooms A-110, A-111, A-113, and A-120 are open.

The order of the delivery steps are that RHINO will first go to A-111 to pick up the book and deliver it in A-113. After the delivery of the book it will pick up the letter and deliver it in the library. This is the schedule that implies the shortest path.

Upon receiving the jobs RHINO starts navigating to Wolfram's desk where the book is loaded (Fig. 3(b)). After RHINO has picked up the book and left room A-111, it notices that room A-113 is closed (Fig. 3(c)). Because RHINO cannot complete the delivery of the book the corresponding command fails. This failure triggers the replanning adaptor **P-3**. Because room A-113 is usually closed it transforms the completion of the delivery into an opportunity. Thus as soon as RHINO notices room A-113 to be open it interrupts its current mission, completes the delivery of the book, and continues with the remaining missions after the book has been successfully delivered.

The new scheduled activity asks RHINO to pick up the letter in A-117 and deliver it in the library. In addition, RHINO receives a third command ("deliver the letter from Oleg's desk in A-119 to Michael's desk in A-120") which is integrated into the current schedule while the schedule is executed. The new order is to get first the letter from A-117 and then the letter from A-119 and to deliver the first letter in A-120 and and then the second in the library. Thus RHINO goes to A-117 to get the letter (Fig. 3(d)) and continues to navigate to room A-119. As it passes room A-113 on its way to A-119 it notices that the door is now open and takes the opportunity to complete the first command (Fig. 3(d)). After that it completes the remaining steps as planned and goes back to its parking position where it waits for new commands (Fig. 3(e-f).

Prediction-based Schedule Debugging. In the third experiment we have shown that strategic adaptations can improve the performance of a robot by performing no worse in situations where foresight is unnecessary and outperforming controllers without strategic adaptations in situations that require foresight. In [BBG99] we have performed experiments with strategical plan adaption processes that perform probabilistic prediction-based schedule debugging (PPSD) for our robot courier application. In one experiment we have tested that PPSDscheduling did on average not worse than situation-based scheduling tactics when no strategic plan adaptations were necessary. In addition, we made up scenarios in which the computation of good schedules required strategic plan adaptations. In those scenarios the controller with a combination of strategic and tactical plan adaptations outperformed the one that only used plan adaptation tactics by about 8% (see [BBG99] for details).

5 Conclusions

In this article we have proposed a new framework for specifying how an autonomous robot should manage the plans it has committed to execute while performing complex jobs in changing environments. The problem in such settings is that the robot's beliefs about the world change continually and therefore the robot must permanently decide whether it should keep executing the plans it has committed to, revise them, or generate and commit to new ones.

The basic idea of our framework is that the plans themselves should specify the plan adaptation processes as subplans. The framework treats plan adaptation actions as computational processes that monitor the belief changes of the agent, and revise the current plan if the processes detect belief changes that warrant a reconsideration of the current course of action. An important advantage of our framework is that all decision making is represented explicitly within the plan and not hidden in a separate system. Furthermore, the interaction between plan adaptation processes and the plans is explicitly specified in the plan. The modular and transparent representation of plan adaptors enables the robot to add, delete, and revise in the same way as other plan steps.

The combination of strategical and tactical plan adaptation provides a powerful framework for the integration of symbolic action planning into autonomous robot control. One of the most attractive features of this approach is that the robot is always under the control of a plan.

Besides the technique itself, the article gives specific examples in which modern AI planning technology can contribute to autonomous robot control by improving the robot's behavior. Strategic plan adaptors can do so because they (1) extend reactive plan execution techniques with means for predicting that the execution of a scheduled activitywill result in a behavior flaw; (2) predict states relevant for making scheduling decisions that the robot won't be able to observe; (3) uses information about predicted states before the robot can observe them.

Michael Beetz has received a Master of Science degree (summa cum laude) from the University of Kaiserslautern and a Master of Philosophy (1994) and Ph.D. degree from Yale University (1996). He is currently a substitute professor in the Computer Science Department of the Technical University Munich. His scientific interests include Artificial Intelligence in general, and plan-based control of autonomous agents in particular.

Contact
Michael Beetz
Technical University Munich,
Department of Computer Science IX,
Orleanstr. 34,
D-81667 Munich, Germany

References

[BBG99] M. Beetz, M. Bennewitz, and H. Grosskreutz. Probabilistic, prediction-based schedule debugging for autonomous robot office couriers. In *Proceedings of the 23rd German Conference on Artificial Intelligence (KI 99), Bonn, Germany*. Springer Verlag, 1999.

[BCF98] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, 1998.

[BDH98] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of AI research*, 1998.

[Bee99] M. Beetz. Structured reactive controllers – a computational model of everyday activity. In O. Etzioni, J. M¨ uller, and J. Bradshaw, editors, *Proceedings of the Third International Conference on Au-tonomous Agents*, pages 228–235, 1999.

[BFG97] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelli-gence*, 9(1), 1997.

[BM96] M. Beetz and D. McDermott. Local planning of ongoing activi-ties. In Brian Drabble, editor, *Third International Conference on AI Planning Systems*, pages 19–26, Morgan Kaufmann, 1996.

[Fir89] J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. Technical report 672, Yale University, Department of Computer Science, 1989.

[IGR92] F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6), 1992.

[McD91] D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[McD92] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[PH99] M. Pollack and J. Horty. There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AI Magazine*, 20(4):71–84, 1999.

[Sch87] M. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1039–1046, Milan, Italy, 1987. Morgan Kaufmann.

[SGH97] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso. Xavier: Experience with a layered robot architecture. *ACM ma gazine* Intelligence, 1997.

[TBB99] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: A second generation mobile tour-guide robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'99)*, 1999.

[Zil96] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.