Plan Representation for Robotic Agents

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

Abstract

Most robotic agents cannot fully exploit plans as resources for better problem-solving performance because of imminent limitations of their plan representations. In this paper we propose plan representations that are, for a given job, representationally and inferentially adequate and inferentially and acquisitionally efficient. We state what these properties mean in the context of robotic agents and describe how plan representations can be designed to satisfy them. The proposed plan representations have been successfully employed in several longterm experiments on autonomous robots.

Introduction

In recent years, robotic agents, including XAVIER (Simmons *et al.* 1997), RHINO (Beetz *et al.* 2001; Burgard *et al.* 2000), MINERVA (Thrun *et al.* 2000), and NMRA (Muscettola *et al.* 1998), have shown impressive performance in longterm demonstrations. These robotic agents have in common that they use plans as resources for improving their competence. Substantial portions of their controllers are implemented as plans, symbolic specifications of the robots' intended activity that cannot only be executed but also reasoned about and revised. Plan-based control enables these robots to flexibly interleave complex and interacting tasks, exploit opportunities, quickly plan their courses of action, and, if necessary, revise their intended activities.

For the design of such control systems it is useful to consider plan languages as a form of knowledge representation. Rich and Knight (1991) propose representational and inferential adequacy and inferential and acquisitional efficiency as key criteria for designing domain knowledge representations. Transferring these notions to plan representation, we consider the representational adequacy of plan representations to be their ability to specify the necessary control patterns and the intentions of the robots. Inferential adequacy is the ability to infer information necessary for dynamically managing, adjusting, and adapting the intended plan during its execution. Inferential efficiency is concerned with the time resources that are required for plan management operations and adaptation. Finally, acquisitional efficiency systems is the degree to which they support the acquisition of new plan schemata and planning knowledge.

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

How well plan languages satisfy these features often determines the degree to which an autonomous robot can

- automatically structure continuous behavior and represent the learned structure as plans that support plan management and opportunistic task execution.
- use and learn routine plans that exhibit coping and situated behavior and have therefore a high expected utility.
- anticipate and forestall execution failures based on realistic predictions of the behavior generated by modern robot control systems.

While these capabilities are necessary for realizing many adaptive robotic agents that are to exhibit competent problem-solving behavior in real environments, surprisingly little effort has been spent on developing plan representations that aim at satisfying combinations of these requirements. This paper reports our experiences in designing special purpose plan representations for robotic agents that aim at being representationally and inferentially adequate as well as inferentially and acquisitionally efficient. Such plan languages provide substantial improvements over existing plan representations in that they support the proper integration of different mechanisms for perception, deliberation, action, learning, and communication and that they provide fast built-in causal and teleological inference and manipulation mechanisms for such expressive plan languages.

We will illustrate the use of such plan languages in the context of designing plans for a robot office courier — in particular navigation plans — and discuss their advantages: First, the high-level plans can achieve better performance than achieved by state-of-the-art navigation systems. Second, the plans can be used in very flexible plans, plans that contain opportunistic plan steps and sensor-driven high-priority tasks. Third, the plans support very fast prediction-based online rescheduling of complex tasks.

In the remainder of this paper we proceed as follows. Section 2 discusses the state-of-the-art in plan-based control of robotic agents. In section 3 we state basic design principles underlying our plan representation. Then we illustrate the principles using example representational constructs and explain our design choices. We conclude with a description of longterm demonstrations and a final discussion.

State of the Art

Current robotic agents typically employ reactive plan languages, problem space languages, and policies for Markov decision problems as their means of behavior specification.

Reactive plan languages are mainly concerned with the effective and competent achievement of goals (Firby 1987; Myers 1996) and not with plan generation and revision. Therefore, the control structures provided by these languages are designed to produce flexible and reliable behavior. They specify coping behavior, situation-specific plan expansion, control of continuous processes, and synchronized concurrent behavior. Coping behavior enables robots to deal with interruptions, small problems, and local failures at execution time. Situated plan execution enables the robots to execute sketchy plans by expanding situation-specific subplans to achieve goals and check whether goals are satisfied upon completion. Situations are used to index the appropriate methods to achieve the goals.

Reactive plans take into account that robot behavior is the result of concurrent, interacting control processes. They change the robot's behavior by changing the set of active control processes. Reactive plans can monitor for important transient situations and handle asynchronous events and signals such as successful goal achievement. Key design issues are synchronization of concurrent processes and the modular design of subplans so that these subplans can be used in multiple constellations of active plans (Firby 1994).

MDP policies model the controlled process as a finite state automaton in which the robot's actions cause stochastic state transitions. The robot is rewarded for achieving its goals quickly and reliably. A solution for such problems is a policy, a mapping from discretized robot states into fine-grained actions. MDPs form an attractive framework for navigation because they use a uniform mechanism for action selection and a parsimonious problem encoding. The policies computed by MDPs aim at robustness and optimizing the average performance. Problems in the application of MDP planning techniques are the size of realistic state spaces and the handling of concurrent actions and dynamically changing and interacting goals.

Problem space plans Most robot action planners make the problem-space hypothesis (Newell 1990), under which problems are stated by a state space and a set of operators that transform states to successor states. A solution is an operator sequence that transforms a given initial state into a state that satisfies the given goal. Problem space plans are designed to simplify plan generation from first principles.

Problem space plans rest on several assumptions. One is the assumption that only temporal orderings cause the interactions between plan steps. Another one is that the robot is always able to complete plan steps. Contrary to these assumptions, many other control patterns for constraining the interactions between plan steps are necessary for flexible and reliable robot control (see above). The problem space plans also do not take into account the interactive character of plan steps, where the plan steps can be initiated by the robot but need outside feedback for their completion.

Problem space plans typically provide only guidelines for execution. This has the disadvantage that planning processes use models that are too abstract for predicting all consequences of their decisions and that planning processes cannot exploit the control structures provided by the lower layer for specifying flexible and reliable behavior.

The lack of sophisticated control structures brought researchers to side step plan representations for the integration of mechanisms others than robot actions. They have specified complex behavior specifications in other formalisms, even when they performed plan management operations. As a consequence, the mechanisms became black boxes for the plan-based control mechanisms. Examples are the situation-specific configuration of image processing routines (Firby *et al.* 1996) and the use of compiled grammars for dialogue control that prevent plan management.

Layered Software Architectures Problem space plans are typically used in layered robot control architectures (Bonasso *et al.* 1997). These architectures run planning and execution at different software layers and different time scales where a sequencing layer synchronizes between both layers. Each layer uses a different form of plan or behavior specification language. The planning layer typically uses a problem space plan, the execution layer employs feedback control routines that can be activated and deactivated. The intermediate layer typically uses a reactive plan language.

While layered architectures mitigate some of the limitations of plan languages they introduce another problem. Instead of a single plan representation the robot has three different ones with incompatible expressiveness. This implies that the representations have to be transformed into each other, which yields considerable information loss.

Requirements

In this paper, we propose the use of a single and application specific plan representations for robotic agents that support (1) a large spectrum of plan management operations, (2) flexible, reliable, and efficient task execution, and (3) the automatic acquisition of robot plans.

Applications of plan-based control. When designing such plan representations it is crucial to look at different applications and their requirements for plan representations.

In our research we design plan-based controllers for robot couriers and tour-guide robots in human environments, autonomous robot soccer, automatic factory control, and distributed supply chain management.

A key characteristic feature of courier and tour-guide applications is that robots have to perform certain tasks at ***specified locations***. The robot courier, for example, has to pick up a letter at one location and deposit it at another one. The robot manages its plan in that it groups its actions according to the locations where they have to be performed. It then sorts the tasks with respect to the locations such that the order satisfies the robots' criterion of utility, for example, to minimize the expected time resources. Thus plan management operations can be supported by plans being modular

and transparent with respect to the locations where the actions have to be performed.

In contrast, in autonomous robot soccer the task of plan-based control is the coordination and synchronizations of the ***movements*** of the different robots. Thus, an appropriate plan representation should represent the movements explicitly, transparently, and modularly. The situation is yet different for the factory application. In factory automation the issue is how to assign jobs to machines, how to ***schedule*** the jobs, and where to place the parts in between the production steps. Finally, in plan-based control of distributed supply chain management the key is to dynamically revise ***negotiation*** tactics.

Properties of plan representation languages. From these considerations we draw the conclusions that a single general plan representation language cannot match all those needs at the same time. Therefore we propose the use of special purpose plan representations which are tailored for the respective application. Let us now flesh out what we mean by representational and inferential adequacy and inferential and acquisitional efficiency in the context of plan-based control of robotic agents.

1. Representational Adequacy: The plans that are reasoned about and manipulated must have the expressiveness of reactive plan languages. In addition to being capable of producing flexible and reliable behavior, the syntactic structure of plans should mirror the control patterns that cause the robot's behavior — they should be realistic models of how the robot achieves its intentions. Plans cannot abstract away from the fact that they generate concurrent, event-driven control processes without the robot losing the capability to predict and forestall many kinds of plan execution failures. A representationally adequate plan representation for robotic agents must also support the control and proper use of the robot's different mechanisms for perception, deliberation, action, and communication. The full exploitation of the robot's different mechanisms requires mechanism-specific control patterns. Control patterns that allow for effective image processing differ from those needed for flexible communication, which in turn differ from those that enable reliable and fast navigation. To fully exploit the robot's different mechanisms, their control must be transparently and explicitly represented as part of the robot's plans. The explicit representation of mechanism control enables the robot to apply the same kinds of planning and learning techniques to all mechanisms and their interaction.

2. Inferential Adequacy: The plan management mechanisms must be equipped with inference techniques that infer the information necessary for plan management. The computational processes for competent plan management must infer the purpose of subplans, find subplans with a particular purpose, automatically generate a plan that can achieve some goal, determine flaws in the behavior that is caused by subplans, and estimate how good the behavior caused by a subplan is with respect to the robot's utility model. Pollack and Horty (1999) stress the point that maintaining an appropriate and working plan requires the robot to perform various kinds of plan management operations including plan generation, plan elaboration, commitment management, environment monitoring, model- and diagnosis-based plan repair, and plan failure prediction.

3. Inferential Efficiency: Plans must support economic inference and plan management. The generation of effective goal-directed behavior in settings where the robots lack perfect knowledge about the environment and the outcomes of actions and environments are complex and dynamic, requires robots to maintain appropriate plans during their activity. They cannot afford to entirely replan their intended course of action every time their beliefs change.

3. Acquisitional Efficiency: Plan representations should support the learning of plans for subsymbolic control processes and efficient and reliable plans for routine activities.

Robot Courier and Navigation Plans

Navigation actions are representative for a large subset of the robots' physical actions: they are movements controlled by the motors of the robot. Physical movements have a number of typical characteristics. First, they are often inaccurate and unreliable. Second, they cause continuous (and sometimes discontinuous) change of the respective part of the robot's state. Third, the interference of concurrent movements can be represented as the superposition of the individual effects. In addition, sometimes movements are planned before they are executed and therefore the causal models need simple models of the plans that are computed. So by designing a plan representation for navigation we hope to get a language that is applicable to a broad class of physical behaviors. The second advantage of choosing navigation as our domain is that navigation is the best understood and developed capability of autonomous mobile robots that can be used for plan-based control.

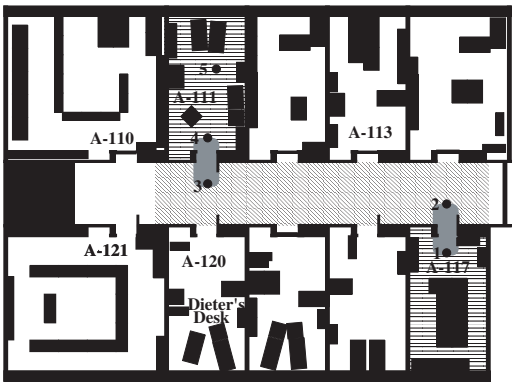

Figure 1: Topological navigation plan for navigating from room A to B with regions indicating different travel modes.

Let us now look at the representational issues for plan-based control of navigation behavior. We will do so by stepwise developing a plan representation for an autonomous robot office courier. We will first represent navigation plans as ordinary reactive plans (3.1). We will then introduce subplan patterns that allow for structuring of continuous control processes and thereby support the acquisition of high-performance symbolic navigation plans (3.2). In the next step, we will take these navigation plans and extend them

such the can be carried out opportunistically or interrupted by opportunistic plan steps (3.3). Finally, we will make the plans that employ these navigation plans more modular and transparent (3.4).

Concurrent Reactive Navigation Plans

Let us first specify navigation behavior using interacting and concurrent control processes. Figure 1 pictures a plan for going to a particular location that consists of two components: the first one causes the robot to follow a sketchy path depicted by the locations indexed through the numbers 1 to 5. The second component specifies when and how the robot is to adapt its travel modes as it follows the navigation path. In many indoor environments it is advantageous to adapt the driving strategy to the surroundings: to drive carefully within offices, to drive with maximal clearance in doorways, and to drive quickly in the hallways. This part of the plan is depicted through regions with different textures for the different travel modes "office," "hallway," and "doorway."

To implement the plan depicted in figure 1 the robot has to detect "events" such as leaving an office and entering a doorway. Therefore, the robot's plans need control structures that respond to signals and events. Useful control structures include *whenever* *s b* that is an endless loop that executes *b* whenever the signal *s* is received and *wait for* (*s*) that blocks a thread of control until signal *s* is received.

The subsequent plan fragment sketches a piece of a navigation plan for leaving the office. The two components following the prescribed path and adapting the travel mode are implemented as concurrent subplans. The second component uses a register to measure the distance to the center of the doorway and two dependent register that signal that the robot enters and leaves the doorway. Initially, the travel mode is set to "office." Upon entering and leaving the doorway the travel mode is adapted.

execute concurrently
  execute-in-order
    GO-TO ***(1)***; *wait for* (***arrived-at(1)?***);
    GO-TO ***(2)***; *wait for* (***arrived-at(2)?***);
  execute-in-order
    SET-NAVIGATION-MODE(office);
    wait for (***entering-doorway?***);
    SET-NAVIGATION-MODE(doorway);
    wait for (***entering-hallway?***);
    SET-NAVIGATION-MODE(hallway)

The first feature that distinguishes our plan representation from many other ones is that it includes all control structures needed to specify reactive plans that produce flexible, robust and coping behavior.

> ***Design rationale 1:*** Provide control abstractions that help to structure complex activities and make them reactive and adaptive. Such control structures include conditionals, loops, program variables, processes, and subroutines as well as high-level constructs (interrupts, monitors) for synchronizing parallel actions.

Why does it help? We have encountered three important reasons why plans of robotic agents should satisfy the design rationale 1. First, sophisticated control patterns are necessary for producing competent behavior. Second, competent plan management requires realistic models of modern robot controllers and the insertion and modification of sophisticated control patterns. Third, to learn plans that produce substantially better performance the plans must specify situated and coping behavior to deal with the high variances real robot behavior.

Structured Reactive Navigation Plans

While concurrent reactive plans are suitable means for hand coding navigation plans, they are very complicated for learning better navigation plans. To remedy these problems we have designed structured reactive navigation plans (SRNPs) that specify a default navigation behavior and use concurrent, percept-driven subplans that overwrite the default behavior while they are active. The (de-)activation conditions of the subplans structure the continuous navigation behavior in a task-specific way. SRNPs have the following form:

navigation plan *(s,d)*
  with subplans *subplan**
  DEFAULT-GO-TO (*d*).

with *subplan*s of the form

SUBPLAN-CALL(*args*)
  parameterizations $\{ p \leftarrow v \}^*$
  path constraints $\{ \langle x,y \rangle \}^*$
  justification *just*

where $p \leftarrow v$ specifies a parameterization of a subsymbolic navigation system. In this expression p is a parameter of the subsymbolic navigation system and v is the value p is to be set to. The parameterization is set when the robot starts executing the respective subplan and reset after the subplan's completion. The path constraints are locations that specify constraints on the robot's path. The subplan call specifies when the subplan starts and when it terminates.

navigation plan *(desk-1,desk-2)*
  with subplans
    TRAVERSE-NARROW-PASSAGE($\langle$635,1274$\rangle$,$\langle$635,1076$\rangle$)
      parameterizations *sonar* $\leftarrow$ *off*
        colli-mode $\leftarrow$ *slow*
      path constraints $\langle$635,1274$\rangle$,$\langle$635,1076$\rangle$
      justification *narrow-passage-bug-3*
      TRAVERSE-NARROW-PASSAGE(...)
      TRAVERSE-FREE-SPACE(...)
  DEFAULT-GO-TO (*desk-2*)

To be more concrete consider the SRNP above, which contains three subplans: two for traversing doorways and one for speeding up the traversal of the hallway. A subplan for leaving an office is shown in more detail. The path constraints are added to the plan for causing the robot to traverse the narrow passage orthogonally with maximal clearance. The parameterizations of the navigation system specify that the robot is asked to drive slowly in the narrow passage.

The plan interpreter expands the navigation subplan into procedural reactive control routines. Roughly, the plan macro expansion does the following. It constructs for the subplan call a monitoring process that signals the activation and termination of the subplan. In our case, the activation and termination of a subplan is triggered by the entering and leaving of rectangular regions in the environment. In addition, the following concurrent process is added to the navigation routine: wait for the activation of the subplan, set the parameterization as specified, wait for the termination of the subplan, and reset the parameterization.

> ***Design rationale 2:*** Provide representational means for structuring continuous control processes using subplans that detect their activation and completion. The activation and completion might be initiated by the continuous effects of the control processes making the activation and termination conditions true.
>
> ***Design rationale 3:*** Use subplan macros to represent common and sophisticated control patterns as a single, compact, and transparent control structure.

Why does it help? The representational means for structuring continuous control processes are important for situated parameterizations of subsymbolic control processes and for learning structured symbolic plans for continuous control processes. The use of subplan macros is important for reducing the search space for transformational learning and for making parameters explicit.

In our example, the subplan macro TRAVERSE-NARROW-PASSAGE is a comprehensive subplan that can be run as a monitoring process and adapts the navigation behavior automatically when the robot traverses the respective narrow passage. All necessary interactions, synchronizations, and local control structures are hidden in the macro definition. The only parts that are transparent are the parameters that are to be reasoned about and adjusted by the learning system. Using this representation the learning algorithm can simply add, delete, and reparameterize the subplans in order to optimize the navigation behavior.

Interruptable and Embeddable Navigation Plans

The navigation plans would be of little use if they could not be employed in diverse task contexts. This can be achieved by turning them into interruptable and embeddable plans.

```
highlevel-plan ACHIEVE loc(⟨x, y⟩)
    with cleanup routine ABORT-NAV-PROCESS do
        with valve wheels do
            loop
                try in parallel
                    wait for(navigation-interrupted?)
                    GENERATE&EXEC-NAV-PLAN(⟨x, y⟩)
                until IS-CLOSE?(⟨x, y⟩)
```

The lines 6 to 8 make the navigation plan independent of its starting position and thereby more general: given a destination d, the plan piece computes a low-level navigation plan from the robot's current location c to d and executes it. To run navigation plans in less constrained task contexts we must prevent other – concurrent – routines from directing the robot to different locations while the navigation plan is executed. We accomplish this by using semaphores or "valves," which can be requested and released. Any plan that asks the robot to move or stand still must request the valve WHEELS, perform its actions only after it has received WHEELS, and release WHEELS after it is done. This is accomplished by the statement *with valve* in line 2.

In many cases processes with higher priorities must move the robot urgently. In this case, blocked valves are simply pre-empted. To make our plan interruptable, robust against such interrupts, the plan has to do two things. First, it has to detect when it gets interrupted and second, it has to handle such interrupts appropriately. This is done by a loop that generates and executes navigation plans for the navigation task until the robot is at its destination. We make the routine cognizant of interruptions by using the register ***navigation-interrupted?***. Interrupts are handled by terminating the current iteration of the loop and starting the next iteration, in which a new navigation plan starting from the robot's new position is generated and executed. Thus, the lines 3-5 make the plan interruptable. To make the navigation plan transparent we name the routine plan **ACHIEVE** $loc(\langle x, y\rangle)$ and thereby enable the plan management component to syntactically infer the purpose of the subplan.

> ***Design rationale 4:*** Use plan libraries with default plan schemata that are general, embeddable, transparent, and interruptable. These properties make it particularly easy to reason about the plans while the plans can still specify the same range of concurrent percept-driven behavior as reactive plan languages can.

A plan p with goal g is general if it accomplishes its goal when confronted with typical situations and events. A plan is embeddable if it achieves its goal even when run with other concurrent and interacting subplans. A plan is interruptable if after non-fatal interruptions by plans the plan will successfully complete the achievement of its goal. A plan is transparent if it is annotated with goal g if and only if the purpose of a plan p is to achieve g.

Why does it help? The plans being *general* helps plan management systems because the systems do not have to reason carefully about the situations in which the plans are to be executed. A plan management system can simply add general plan steps to a plan, without worrying exactly where, and can still expect the plan step to have the desired effects. In order to recover from local execution failures the system can simply add a general plan step with the needed effects or call the general plans repeatedly.

For example, consider the classical block stacking problem. It can be solved by having a general plan for an individual stacking task, performing the necessary stacking steps in a random order, and iterating this step until all individual stacking tasks are achieved. This works because in each iteration at least one additional stacking goal is achieved. Generality ensures that plan steps do what they are supposed to. *Embeddability* ensures that an added or rearranged plan step does not cause negative interferences in other plan steps. Taken together, generality and embeddability enable

the plan management system to place subplans much more sloppily. This is very important because concurrent reactive plans are very complex and for execution time plan management the robot often does not have the time resources to reason through all consequences of its operations.

Delivery tour plans

To facilitate online rescheduling we have modularized the plans wrt. the locations where subplans are to be executed (McDermott 1992). The *at location* ⟨x,y⟩ *p* plan schema 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 for Task-Interrupted?
                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)*.

```
partial order
      named subplan s_1
        if EXEC-STATE(p,to-be-acquired)
          then at location L PICK-UP(p)
      named subplan s_2
        if EXEC-STATE(p,loaded)
          then at location D UNLOAD(p)
```

If every subplan *p* that is to be performed at a particular location l has the form *at location* ⟨x,y⟩ *p*, then a scheduler can traverse the plan recursively and collect the at location subplans and install additional ordering constraints on these subplans to maximize the plan's expected utility. To allow for smooth integration of revisions into ongoing scheduled activities we designed the plans such 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 made the plans for single deliveries 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.

> ***Design rationale 5:*** Use modular and declarative subplans/subplan macros genuinely.

Why does it help? The use of subplan macros allows for very fast access of plan steps through plan management operations and the modularity achieved by the macros makes plan revision robust. For example, using the at location macro we can make a plan location transparent, by wrapping every subplan *p* that is to be performed at a particular location ⟨x,y⟩ with *at location* ⟨x,y⟩ *p*. 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 computes and installs an extension of these orderings that maximizes the expected utility.

In addition, declarative subplan macros discharge the plan management from performing time intensive reasoning tasks. If for a state of affairs *p* the robot has a means for testing whether *p* holds and general, embeddable plans that achieve *p*, maintain *p*, prevent *p*, and compute its belief state with respect to *p* then a number of important plan management operations can be realized easily. It can test whether a plan *achieve (p)* has been successful, it can monitor the state of affairs *p* by simply adding a plan step *maintain (belief-state(p))*, or it can *maintain (p)* over a critical interval. Having transparent plans such general plan transformations are much easier to write.

Consider the robot location as an example state. We can make the robot cognizant about dealing with locations by providing a plan schema for achieving *p* that simply navigates to the respective location, a schema for maintaining *p* by going back to the location whenever the robot has left it (e.g., in order to recharge batteries). The belief state with respect to its location can be computed through a self localization method.

Self-adapting Plans

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 plan piece

```
With-plan-adaptor Whenever the robot receives new requests
                    if its set of Jobs J might interfere
                      then it changes the course of action
                           to avoid these interferences
```

Concurrent reactive delivery plan

tells the robot to carry out *delivery plan* but if it receives new requests and the achievement of requests might interfere then the plan adaptor is to change the course of action to

```
top-level-plan OFFICECOURIER( )
 1  with stabilizer relocalize when necessary
 2                  reload battery when necessary
 3  do with plan adaptor manage the plan with respect to the user's requests
 4                       and the robot's utility model
 5     do named subplan main − plan
 6     do with stabilizer situation specific stabilizers
 7        do with plan adaptor situation specific plan adaptors
 8           do partial order set of delivery plans
 9                          containing at location subplans
10              :order ordering constraints on at location subplans
```

Figure 2: Top-level plan of the robot office courier

avoid the interferences. Clearly, the plan adaptor does planning: it predicts the consequences of inserting new requests and changes the plan in response to these predictions.

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

> ***Design rationale 6:*** represent plan management routines explicitly.

Why does it help? Representing plan management routines explicitly as parts of the plan enables us to control and synchronize the routines in the same way as the physical actions of the robot.

The Top-level Plan of the Robot Courier

Figure 2 sketches the top-level plan of the robot office courier. The intended delivery tour (lines 8-10) is specified by a set of *at location* subplans and ordering constraints. This course of action is constrained by stabilizers and plan adaptors. Stabilizers are monitoring subplans that restore conditions that are necessary for the successful completion of the deliveries. These conditions include that the robot keeps its battery loaded and relocalizies itself whenever it has lost track of its position. The outer plan adaptors make sure that the intended course of action contains plan steps for all requests that are currently achieveable. The inner plan adaptors are installed and retracted by the outer ones and perform temporary plan management tasks such as computing plans for completing locally failed subplans.

The Plan Representation Language

So far we have discussed representational concepts but not the plan representation language itself. Let us now turn to the issue of how to define the plan representation language. An obvious choice is a syntactic definition of the plan representation as the transitive closure of the primitive plans with respect to its control structures. Unfortunately, such a definition yields plan languages for which we cannot provide reliable and fast algorithms for plan synthesis, failure prediction and diagnosis, and execution time plan revisions.

So let us consider an alternative definition of the plan language. Many robotic agents are provided with a library of modular default plans that are efficient and cope with most eventualities. Using such a library, the robot computes a default plan for a set of tasks by retrieving and instantiating plans for individual tasks and pasting them together to form a parallel robot plan. All other plans result from performing plan management operations on these plans.

This suggests an operational definition of the plan language as the set of plans that results from the application of the plan management operations to the basic plans. This definition is much more attractive because it enables robotic agents to simplify the computational problems by making assumptions about plans instead of making assumptions about worlds and robots, such as the robot being omniscient, the world being static, etc. The planner constructs and revises the plans and can thereby enforce that assumptions that simplify the inference problems hold.

> ***Design rationale 7:*** define plan languages operationally.

Why does it help? An operational definition of the plan language provides us with an effective means for achieving that all plans are *restartable*, *embeddable*, *general*, and *transparent*. In our applications we could produce each desired behavior using a plan schema that satisfies these properties. Thus requiring these properties for plans does not limit the expressiveness of the behavior specification language. We have also designed the plan management operations such that they preserve these properties. Thereby we can be sure that all our plans are *restartable*, *embeddable*, *general*, and *transparent* and can therefore apply plan management algorithms that exploit these properties and can therefore be much faster and more reliable than their counterparts that cannot make these assumptions.

Demonstrations

The proposed plan representation has been employed in several demonstrations of robotic agents. In one of them a plan-based controller has controlled the museums tourguide robot MINERVA that has operated for a period of thirteen