Improving Robot Plans During Their Execution*

Michael Beetz and Drew McDermott
Yale University, Department of Computer Science
P.O. Box 2158, Yale Station
New Haven, CT 06520
beetz@cs.yale.edu, mcdermott@cs.yale.edu

Abstract

We describe how our planner, XFRM, carries out the process of anticipating and forestalling execution failures. XFRM is a planning system that is embedded in a simulated robot performing a varying set of complex tasks in a changing and partially unknown environment. XFRM revises plans controlling the robot while they are executed. Thus whenever the robot detects a contingency, XFRM projects the effects of the contingency on its plan and—if necessary—revises its plan in order to make it more robust. Using XFRM, the robot can perform its tasks almost as efficiently as it could using efficient default plans, but much more robustly. Revising default plans requires XFRM to reason about full-fledged robot plans and diagnose various kinds of plan failures that might be caused by imperfect sensing and effecting, incomplete and faulty world models, and exogenous events. To this end, XFRM reasons about the structure, function, and behavior of plans, and diagnoses projected plan failures by classifying them in a taxonomy of predefined failure models. Declarative commands for goals, perceptions, and beliefs make the structure of robot plans and the functions of subplans explicit and thereby provide XFRM with a (partial) model of its plan that is used to perform hierarchical model-based diagnosis.

Introduction

A very common idea in the planning literature is the idea of a *plan library*, a collection of canned plans for achieving standard goals in standard situations. The advantage of a plan library is that it can generate plans quickly, which is crucial in dynamic, imprecisely known environments. However, in such an environment it is hard to be sure that a plan is anywhere near optimal, especially a canned plan. For example, if a plan is to bring about some state of affairs—a *goal*· in such a way that the state is likely to last for a while, then the amount of effort to expend on making sure the state is stable depends on the expected interferences. If the goal is to keep an object at a certain location, then it is worthwhile nailing it down only if there is reason to believe that some other agent might move it. A canned reactive plan will have trouble testing for such an expectation, and so the decision to stabilize the goal must typically be made once and for all when the plan library is built.

An alternative is to equip a robot with a planning system that continually optimizes a default plan for the current constellation of goals based on the robot's observations of its environment. This is the job of XFRM, the planning system described in this paper. XFRM is embedded in a simulated robot equipped with limited sensing and effecting capabilities. The robot performs a varying set of tasks in a changing and partially unknown environment, a small world with various objects to be transported from place to place. The plans controlling our robot are made robust by incorporating sensing and monitoring actions, and reactions triggered by observed events. XFRM is provided with a library of modular default plans that are efficient and cope with most eventualities. Using this library, XFRM 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. While pasting default plans together is fast, it is prone to producing plans that may fail for contingencies. Hence, whenever the robot detects a contingency, XFRM will project the effects of this contingency on its current plan and revise the plan to make it more robust. Whenever XFRM thinks it has a plan that is better than the currently executed one, it will replace the current plan with the better one and restart the new plan.

Planning is implemented as a search in plan space. A node in the space is a proposed plan; the initial node is the default plan created using the plan library. A step in the space requires three phases. First, XFRM *projects* a plan to generate sample execution scenarios for it. Then, in the *criticism* phase, XFRM examines these execution scenarios to estimate how good the plan is and to predict possible plan failures. It diagnoses the projected plan failures by classifying them in a taxonomy of failure models. The failure models serve as indices into a set of transformation rules that

*This work was supported by the Defense Advanced Research Projects Agency, contract number N00014-91-J-1577, administered by the Office of Naval Research

are applied in the third phase, *revision*, to produce new versions of the plan that are, we hope, improvements.

The following example illustrates how XFRM is supposed to work. Suppose, one top-level command asks the robot to get all balls from location ⟨0,8⟩ to location ⟨0,10⟩. XFRM retrieves a default plan for carrying out this task: go to ⟨0,8⟩, find a ball there, deliver it to ⟨0,10⟩, and keep doing this until there is no ball left at ⟨0,8⟩. During the execution of this routine, the robot notices a cleaning robot at ⟨0,9⟩. While the control system continues to execute the default plan, XFRM projects the effects of its current plan with the cleaning robot working at ⟨0,8⟩. Among other things, the probabilistic model of the cleaning robot specifies, that it often moves objects to adjacent locations, and often makes objects wet. Now suppose that in one of the following, probabilistically projected, execution scenarios, XFRM predicts the other robot to clean ⟨0,8⟩. In this execution scenario the robot is also predicted not to deliver all balls from ⟨0,8⟩ to ⟨0,10⟩. In one of these cases, the subplan for perceiving balls couldn't recognize an object as a ball, because the object recognition algorithm couldn't deal with specular reflections of wet objects. Another ball was not delivered because it was moved to a neighboring location. To fix the first failure, the robot can substitute a more reliable and expensive vision processor. A fix for the second failure is to mark all the balls that are to be delivered in the beginning. If a marked ball is missing the robot looks for it at a neighbor location.

In the remainder of the paper we discuss how XFRM predicts the effects of contingencies on its plan and revises its plan to make it robust against these effects. Here, we focus on *local* transformations, which make a plan step more reliable by making changes only to that step. There is a broader class of "global transformations," which can affect the entire plan. An example would be dealing with the cleaning robot by constraining plan steps to precede or follow its projected actions. See (McDermott 1992) for examples of global transformations.

Plan Representation in XFRM

Currently, XFRM does errand planning in a simulated world of discrete locations. Boxes, blocks, and balls with different colors, textures, and sizes, are spread out in the world and can move, appear, disappear, or change their appearance due to exogenous events. XFRM controls a robot with two hands and two cameras. By using a camera the robot can look for objects at its current location, track objects in the camera view, or examine a tracked object. Sensor programs look for objects matching a given description and return a *designator* describing each matching object and its location. Since the robot's cameras are imperfect, designators will be inaccurate and the robot may overlook or hallucinate objects. To change its environment, the robot can reach out a hand to the positions of tracked objects, grasp them, and thereby pick them up. While grasping or carrying objects, objects might slip out of the robot's hands or might be stuck when the robot opens a hand. This environment plays the same role as the "Tileword" family of world simulators.[1] We hope it is slightly more realistic.

While acting in its environment, the robot stores and updates designators and uses them to find and manipulate the objects they describe. The known designators constitute the robot's model of the current state of its environment. This model is necessarily incomplete because it only contains designators for already perceived objects. It may even be wrong since (a) the designators are produced by noisy sensors and (b) exogenous events might have changed the world without the robot noticing. These deficiencies of its world model imply that XFRM cannot make all control decisions in advance: To get an unknown red object from another location, the robot has to get there and find a red object before it can decide how to grasp it. Also, the robot has to watch out for exogenous events and to react appropriately so that they do not interfere with its mission. Thus control routines that are robust for our environment require parallelism, complex flow control, and the capability to store information. Rather than hiding this complexity from XFRM we encode the whole plan in RPL *(Reactive Plan Language)* (McDermott 1991), a LISP-like robot control language with conditionals, loops, program variables, processes, and subroutines. RPL also provides high-level constructs (interrupts, monitors) to synchronize parallel actions and make plans reactive and robust.

Besides constructs for reactive robot control, RPL offers declarative statements, which allow for the representation of goals, perceptual descriptions of objects, and beliefs as logical expressions (Beetz & McDermott 1992): For instance, (ACHIEVE *(LOC DES* (0,10)*)*) specifies the task to get the object denoted by the designator *DES* to the location (0,10). The perception statement (PERCEIVE *D*) gets a perceptual description *D* and returns designators for objects satisfying *D*. The statement (BELIEF *(IN-HAND DES H)*) makes the data structure HAND-MODEL, which is used by our robot plans, transparent. (BELIEF *(IN-HAND DES H)*) is true if the value of the slot CONTENT of hand *H* is *DES*, i.e., if the robot "believes" that it has the object designated by *DES* in its hand *H*. Declarative statements make the purpose of subplans explicit.

In order to execute declarative statements the robot controller needs plans for achieving the goals or making the perceptions specified by the statements. These routines are stored as procedure definitions in XFRM's plan library. The entries in XFRM's plan library have been carefully designed to work for most situations encountered in our environment, and to be able to recover from common plan failures. A key constraint on

[1] See (Hanks, Pollack, & Cohen 1993) for a discussion of the use and limitations of simulators for agent research.

the design of plans is that they are *restartable*. As explained earlier, the agent will swap a new plan in whenever it thinks it has a better version. Hence no plan's correctness should depend on saved state; starting over again should always work. For details, see (McDermott 1992).

Projected Execution Scenarios

While the robot performs its tasks as specified by the default plan that is glued together from the entries in the plan library, XFRM projects this plan in order to generate and analyze possible execution scenarios for it. A projected execution scenario is represented by a task network and a timeline. The *task network* is a tree of tasks and their subtasks (that is, activation records for every piece of the plan for the top task in the tree). For each task and subtask the projector stores its outcome (success, failure, ...) and how it was attempted. The *timeline* is a sequence of timepoints, where each timepoint is the start or end of a robot action, or an external event. It records how the world evolved in the projected execution scenario. Projection is probabilistic, so it is necessary to generate multiple scenarios to be sure that a moderately probable bug will manifest itself. (For details, see (McDermott 1994).)

XFRM diagnoses plan failures by analyzing projected execution scenarios. Diagnostic rules are written in XFRM-ML, a PROLOG-like language that comprises PROLOG primitives, a LISP interface, and a set of built-in predicates for temporal reasoning, as well as predicates on task networks and RPL plans. These built-in predicates reconstruct—when necessary—the state of the robot and its environment, the status of tasks (succeeded, failed, active, etc.), and the value of program variables and data structures at arbitrary points in a projected scenario. A detailed description of the algorithms for projecting parallel RPL plans, the language for describing causal models of actions and events, and the mechanisms for reconstructing a RPL program's and robot's state after a projection can be found in (McDermott 1992). Given a projected execution scenario, XFRM's critics can infer via XFRM-ML answers to queries like: Was the task (ACHIEVE *G*) carried out successfully? Did the robot try to achieve *G*? Did the robot believe the environment to be in a state *S* at a given stage of the projected plan execution? Do the designators returned by the task (PERCEIVE-ALL *D*) answer the robot's information requests correctly and completely?

Thus XFRM may ask whether the robot has tried to achieve a goal using the query (DECLARATIVE-SUBTASK *ST* (ACHIEVE *G*) *TSK PROJ*), which succeeds if *ST* is a subtask of the task *TSK* in the projection *PROJ* and the code of *ST* matches (ACHIEVE *G*). The value *VAL* of a given RPL expression *EXP* before a task *TSK* can be retrieved or verified by (VALUE *EXP TSK* BEFORE *PROJ VAL*). Several predicates are provided to query timelines. (HOLDS-TT-TASK *S TSK PROJ*) succeeds if the state *S* holds throughout the task *TSK*. Similarly, we have predicates for checking states before, during, and after a task. Other predicates defined for timelines determine what happened at a given timepoint or enumerate the timepoints in a timeline. XFRM-ML predicates defined on plan text comprise (RPL-EXP *EXP CODE*), which holds if *EXP* is the plan text executed in order to perform task *TSK*. (ABSTRACT-RPL-EXP *REDUCEE EXP*) does the same but also walks backward through plan expansion and procedure calls.

Diagnosis of Projected Plan Failures

There are two kinds of plan failure: plan-generated, and inferred. A *plan-generated* failure is signalled by the plan itself during execution or projection, by use of the RPL FAIL construct. An *inferred* plan failure is deduced by the planner by comparison of what was supposed to happen with what actually happened during execution or projection. Such an inference is possible only if the planner has models of what was supposed to happen and what actually did. Each declarative construct supplies a model of the first kind, which we call the *behavior specification* of the construct. The behavior specification for a task of the form (ACHIEVE-FOR-ALL *D G*), for instance, specifies that all objects satisfying *D* have to satisfy *G* at the end of the task. The second model, of what really "happened," is supplied by the projector. Deciding what happened from a projection is a matter of retrieval and inference. Deciding what really happened after an actual execution might require further planning and acting, and we will have no more to say about it here.[2]

Behavior specifications of declarative constructs are formalized as XFRM-ML Horn-clause rules for the predicate (UNACHIEVED-TASK-BUGS *TSK E-SC BUGS*), which is true if *BUGS* describe all violations of the behavior specification of *TSK* in the execution scenario *E-SC*. When the query succeeds, *BUGS* is bound to a set of failure descriptions that name a top-level task and the subgoals of its behavior specification that haven't been achieved. Applying appropriate plan revision methods to the relevant subplans, however, requires XFRM to know the subplans that cause a detected failure and the reason why the failure occurred. Therefore, XFRM diagnoses a failure by classifying it, using a taxonomy of failure models, and produces a detailed failure description, which contains pointers to appropriate plan revision methods and the subplans that have to be revised. (This idea is due to (Sussman 1977).)

Figure 1 shows part of the failure taxonomy. It classifies failures as caused by *imperfect sensors*, *imper-*

[2]See (McDermott 1992) for more on plan-generated failures that occur during projection. We have not yet devoted any special attention to the case of failures that occur during execution. At present, the planner cannot infer a failure after the execution of a plan; it deals with plan-generated failures by simply restarting from the new state of affairs that obtains after the failure.

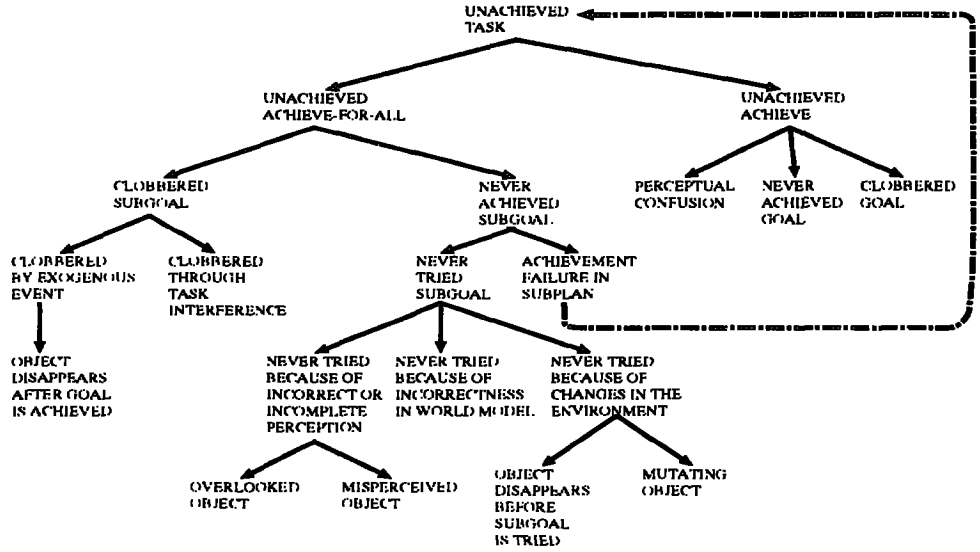

Figure 1: Part of XFRM's taxonomy of failure models.

fect control, an *inaccurate and incomplete world model*, and *other tasks*. Each failure model has a definition, which is a XFRM-ML clause that is true if and only if the given failure satisfies the model. The failure classification starts with the failure description produced by the behavior specifications and classifies the failure first with respect to the command name of the top-level command in which the failure occurred. A task of the form (ACHIEVE-FOR-ALL D G) can fail to achieve a subgoal in two ways. The first possibility is that the plan never achieved the goal G (*never achieved subgoal*), the second that G has been achieved but ceased to hold before the task's end (*clobbered goal*). The query (HOLDS-DURING-TASK *?G ?TSK ?PROJ*) determines which of both failures occurred. If a subgoal was never achieved, cases in which it was never tried (*never tried subgoal*) and others in which the robot failed to achieve it (*achievement failure*) are distinguished. The criterion for differentiating between them is whether or not the declarative subtask (ACHIEVE G) is found in the task network. When an already achieved goal has been clobbered, it is useful to distinguish whether the cause is a robot action (*task interference*) or an *exogenous event*. In the first case the planner can set up protections, apply alternative goal reductions, or add ordering constraints on subtasks to avoid the failures. If the achievement of G failed (*achievement failure*), XFRM would try to identify subtasks that didn't work, classify their failures, and return the resulting diagnoses.

The taxonomy of figure 1 is moderately domain-dependent. Some parts of it would carry over unchanged to other environments with similar degrees of agent control. On the other hand, a case can be made that the taxonomy ought to be more carefully tailored for each environment the agent can operate in, much as the plan library is tailored. The taxonomy is complete in that any projected failure of an ACHIEVE or ACHIEVE-FOR-ALL is described by failure models.

The Revision of RPL Plans

For each failure model, we store a (non-empty) set of plan-revision methods that can make a plan more robust against failures of this type. Plan-revision methods check their applicability, recompute, if necessary, which parts of the plan text need to be revised, and edit the plan to perform the revision. Because our focus in this paper is on local transformations, we can implement plan transformations using rules of the form $\frac{ips}{ops}[ac$, where ac is the applicability condition. *ips* the input plan fragment and *ops* the output plan fragment of the rule. The applicability condition is a conjunction of XFRM-ML clauses. The input plan schema is a pattern for a RPL plan that is matched against the subplan to be revised. *ops* is a RPL plan fragment containing pattern variables that get instantiated using the bindings produced by proving the applicability condition and matching the input plan schema. The resulting plan fragment replaces the input fragment in the revised plan. The example in figure 2 replaces a failing plan step *?DELIVERY-PLAN* with a sequence (SEQ) consisting of *?DELIVERY-PLAN* followed by nailing down the delivered object *?VAR*; it does this under a set of conditions described on the right side of the figure. These conditions specify that (a) The unachieved subgoal is a delivery, i.e., a goal of the form (*LOC ?OB* (X.Y)); (b) The robot has tried to deliver an object denoted by a designator *?D* where *?D* referred to *?OB* (subtask *?AT*); and (c) The environment of the sub-

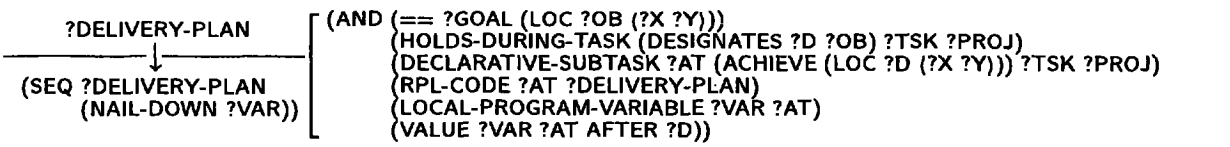

Figure 2: One of the plan transformation rules that is stored for the failure model *object disappears after subgoal is achieved.*

plan *?DELIVERY-PLAN* contains a local program variable *?VAR* that has the value *?D* at the end of task *?AT*. The rule revises *?DELIVERY-PLAN*, the plan for carrying out *?AT*.

Several plan transformation rules are stored under the failure model *"object disappears after subgoal is achieved"*. XFRM tries the transformations in depth-first order. One of these revisions is to monitor an already achieved goal G, in our case the location of a delivered object, and to reachieve G whenever it ceases to hold. Another revision is to bring the objects into a state in which they cannot disappear. In our domain objects cannot disappear if they are nailed to the floor or inside a box. Since it costs time to nail objects to the floor and since the nails have to be removed before the robot can manipulate the objects anew, the robot does not nail down objects by default. XFRM will revise the default plans to nail down objects only if it has reason to believe that delivered objects might disappear.

A plan transformation is not guaranteed to improve the plan, or even eliminate the bug that prompted it. Consider a plan for achieving G that requires a state P to succeed. The revision that asks to monitor P while achieving G and reachieving P whenever it ceases to hold seems to be more robust than (ACHIEVE G) because it maintains P while achieving G. However, the monitor could overlook cases in which P became false, or it could signal that P became false if it didn't. Also the control program for reachieving P might fail. The best XFRM can do is to estimate whether a plan revision is likely to make a plan more robust by analyzing execution scenarios for the resulting revision.

Experiment

In order to study the feasibility of our approach we have performed the following experiment: The robot is at ⟨0,0⟩ and has to get all balls from ⟨0,8⟩ to ⟨0,10⟩. While carrying out the task, the following contingencies may or may not occur: balls may disappear or change their shape, sensor programs may overlook objects or examine them incorrectly. We assume that XFRM's projector has a good model of the current probabilities of the contingencies. (Imagine that the probability is calculated based on the last sighting of the "cleaning robot," although this experiment does not actually involve such an autonomous agent.)

One approach to this situation would be to employ as a default a plan, which we'll call the *ultracautious* plan for a specific contingency c, that always acts as if interference caused by c is probable. For the contingency that objects disappear from location ⟨0,10⟩, the ultracautious plan will nail the balls down immediately after their delivery. In the case that balls might change their shapes, the ultracautious plan will mark each ball and then deliver each marked object. But if the planner has knowledge that interference is not currently likely, the ultracautious plan will do a lot of unnecessary work. Hence we use as our default a version of the plan we'll call the (merely) *careful* plan. This plan deals with common problems like balls slipping out of the robot's gripper by trying to pick up objects repeatedly until the grasp is successful. Also, the careful plan monitors the hand force sensor during the delivery. If the hand force drops to zero, the plan asks the robot to look for the lost ball, to pick it up again, and to continue its delivery. In addition, XFRM is provided with transformations that allow it to revise the careful plan when interference is predicted.

command	time	goals	utility	time	goals	utility
	balls disappear			balls don't disappear		
careful	241	2/4	159.8	237	4/4	360.5
ultracaut.	323	4/4	306.1	325	4/4	305.8
planned	347	4/4	302.1	248	4/4	358.6
	balls change their shape			balls don't change		
careful	153	2/4	174.5	246	4/4	359.0
ultracaut.	315	4/4	347.5	305	4/4	349.2
planned	325	4/4	345.8	251	4/4	358.2
	balls are overlooked			balls aren't overlooked		
careful	200	3/4	266.6	240	4/4	360.0
planned	255	4/4	357.5	252	4/4	358.0

Figure 3: Comparison between careful default plan, ultracautious plan, and planning simultaneous with execution of the careful default plan. Duration is the time needed to perform the task in seconds. achieved goals specifies the number of goals that have been achieved. Benefit is the amount of "utility" earned by the robot, where 10 minutes execution time cost 100 units, a delivered ball brings 100 units if it is not nailed down, 90 units otherwise.

For each contingency c we compare the performance of three robot controllers: (1) the careful (default) plan, (2) the ultracautious plan, and (3) a controller that starts with the careful plan, but runs XFRM to improve it as it is executed. As execution proceeds, XFRM projects the default plan, revises it, if necessary, and swaps in the revised plan. As we said, we assume that XFRM is given a causal model of c that predicts fairly

well whether or not c occurs. We run each controller in two scenarios: in one the contingency c occurs and in the other it doesn't.

Figure 3 shows the trials for four contingencies. As expected, the ultracautious plans are significantly slower than the careful plan but achieve all their subgoals whether or not c occurs. The careful plan works well only if c does not occur; if c does occur, some subgoals are not achieved successfully. The controller with planning is as robust as the ultracautious plan. It is also on average 12.8 seconds (less than 5%) slower than the executing the best plan that achieves all the goals on a given trial. Planning costs are this low because XFRM plans during the idle time in which the interpreter was waiting for feedback from control routines. In terms of average benefit, the controller assisted by XFRM was better (337.0) than the ultracautious (321.0) and the careful plan (261.7). We should interpret the experimental results carefully: XFRM had a very good model of contingencies.[3] However, we believe that the experiment shows that a robot controller assisted by XFRM can outperform robot controllers that do not plan and be almost as efficient as a system that guesses the best plan to use on each occasion.

It might seem that there is an alternative we did not test, namely, to write a careful reactive plan that tests its world model once, then switches to an ultracautious version if interference is anticipated. There are two main reasons to reject this approach: (1) It is not clear what to test; the interference may be the result of several interacting processes that emerge in the projection, so that the simplest test might just *be* the projection itself. (2) The transformation rules are not specific to a particular plan. Rather than write alternative versions of every plan, we can write a single transformation that can revise a variety of plans to compensate for certain standard bugs.

Conclusion

Our preliminary conclusion is that we can avoid the consequences of unusual but predictable adverse circumstances by predicting them and transforming the plan to be robust against them. Using this approach, we can implement robot controllers that achieve their tasks almost as efficiently as, but much more robustly than, canned reactive plans. XFRM works because the reactive robot plans provided are designed such that XFRM can reason about the structure, functions, and behaviors of plans. Secondly, XFRM has a taxonomy of failure models, defined as logical expressions, for which XFRM can check whether or not they are satisfied by the behavior of a projected task.

XFRM, like HACKER (Sussman 1977), GORDIUS (Simmons 1992), and CHEF (Hammond 1989), is a transformational planner that diagnoses "bugs" or, in our case, plan failures, in order to revise plans appropriately. XFRM differs from other transformational planners in that it tries to improve a simultaneously executed plan instead of constructing a correct plan. Also, XFRM reasons about full-fledged robot plans and is able to diagnose a larger variety of failures. GORDIUS also applies model-based diagnosis methods by propagating causal constraints through a partially ordered set of plan steps. We use hierarchical model-based diagnosis with explicit failure models (Hamscher 1991) since the RPL plan steps and their effects are typically modified by parallel monitoring processes and the control structures the step occurs in. Other planning architectures that make plans more robust during their execution mainly address plan failures caused by nondeterministic outcomes of plan actions (Drummond & Bresina 1990, Dean *et al.* 1993), or they plan for single, highly repetitive tasks (Lyons & Hendriks 1992).

Acknowledgments. We would like to thank Sean Engelson, Sam Steel, Wenhong Zhu, and the reviewers for valuable comments on earlier versions of this paper.

References

Beetz, M., and McDermott, D. 1992. Declarative goals in reactive plans. In Hendler, J., ed., *AIPS-92*, 3–12.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proc. of AAAI-93*, 574–579.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. of AAAI-90*, 138–144.

Hammond, K. J. 1989. *Case-Based Planning.* Academic Press, Inc.

Hamscher, W. C. 1991. Modeling digital circuits for troubleshooting. *Artificial Intelligence* 51:223–271.

Hanks, S.; Pollack, M.; and Cohen, P. 1993. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine* 14(4):17–42.

Lyons, D., and Hendriks, A. 1992. A practical approach to integrating reaction and deliberation. In Hendler, J., ed., *AIPS-92*, 153–162.

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

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

McDermott, D. 1994. An algorithm for probabilistic, totally-ordered temporal projection. Research Report YALEU/DCS/RR-1014, Yale University, Department of Computer Science.

Simmons, R. G. 1992. The roles of associational and causal reasoning in problem solving. *Artificial Intelligence* 53:159–207.

Sussman, G. J. 1977. *A Computer Model of Skill Acquisition*, volume 1 of *Aritficial Intelligence Series*. New York, NY: American Elsevier.

[3] We haven't studied how XFRM might learn such a model from experience, or what the penalties are for not having learned it perfectly.