Execution Monitoring

– A Survey –

Fall 2005

Christian Fritz
`fritz@cs.toronto.edu`

University of Toronto
Department of Computer Science
Toronto, Ontario, Canada.

Contents

1 Introduction

In this document we survey the existing work on the problem of monitoring the execution of agent plans and reacting to unexpected events (e.g. precondition failures). We start off by motivating the problem and presenting our view of the structure of the problem. Guided by this structure we then review the literature relevant to specific aspects of the problem, discussing their merits and limitations, and identify open problems. The problem of execution monitoring has attained comparatively little attention and there is in particular very little principled work on this topic. Hence, this survey mainly gives an overview of the available systems and makes an attempt to collect and generalize the contained ideas. Before concluding, we summarize our observations and list possible research opportunities based on the ideas and limitations identified.

1.1 Motivation

In order to make an agent reach a specified goal, it is common to device an agent with the ability to plan. When there are several ways of reaching the goal, there are several *valid* plans, one typically prefers one over the others leading to the notion of an *optimal* plan. Independent of the underlying planning paradigm, algorithm, and preference specification there are always assumptions about the world involved, as any kind of planning has to apply a model of the world dynamics and in particular the agent's action's impact on the state of the world in order to produce valid and optimal plans. But these assumptions may turn out to be wrong or imprecise causing the world to evolve differently than expected. In 1977, McCarthy [McCarthy, 1977] identified the so-called *qualification problem*, the impossibility of stating and reasoning with all qualifications, i.e. preconditions, under which an action in the domain of interest has its intended effects. This gives rise to potential modeling errors that may result in discrepancies between the actual world state and the predictions thereof.

These discrepancies may or may not affect the validity and optimality of the plan being executed but if they do, replanning of some sort is required to maintain these properties. Other sources of discrepancies could be unexpected exogenous events or, more subtle, a change of objective, e.g. a different goal.

As a result, agents acting in this kind of domain should be aware of the possibility that a laid out plan may be rendered sub-optimal or invalid over the time it is being executed, and provide for means of rationally reacting to these circumstances. We call the combination of verifying the continued validity or optimality of a plan being executed (monitoring) and the possibly required replanning in case sub-optimality is asserted *execution monitoring*.

In the next section we present our understanding of the overall execution monitoring framework, dividing the problem into sub-problems. The sections thereafter will be structured according to these sub-problems.

2 Framework

In this section we describe a possible subdivision of the overall task of execution monitoring. The choice of sub-tasks is inspired by the reviewed literature and previous framework specifications, which we discuss at the end of this section. We distinguish a monitoring and a replanning module, where the monitoring module is subdivided into a discrepancy detection step comparing expectations and observations, a diagnosis or state estimation step, and an evaluation step. This structure is depicted in Figure 1. Since we are going to structure our

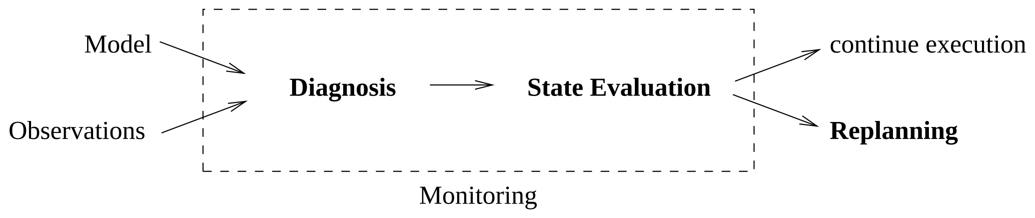

Figure 1: Our execution monitoring framework.

presentation of the relevant literature accordingly, we briefly describe these components.

Model: This is the model of the world used in planning.

Observations: We are assuming a situated agent, that is, an agent performing actions in an environment, thereby causing changes therein and receiving sensory input from this environment. This input can appear in response to active sensing requests or be delivered to the passive agent at a certain frequency. Typically, these observations do not reveal the complete state of the world.

Diagnosis: This is the task of estimating the actual state and determining a sequence of events that produced it, based on the model and the observation history. Which aspects of the current state one aims to estimate is determined by the subsequent steps, Evaluation and Replanning. In the context of execution monitoring, state estimation is typically tailored towards detecting the discrepancies between what was predicted by the model and what was actually observed in the real world. We are going to study this task in detail in Section 4.3.1.

State Evaluation: This is the task of determining whether in the estimated state the validity and/or optimality of the current plan is preserved or not. If it is not, this step leads to replanning, otherwise the current plan is continued. In the artificial example domains found in the planning literature generally only those features of the world that matter to the problem are modeled. Then virtually any discrepancy between the expected and actual state of the world matters, i.e. affects plan validity and/or optimality. But this is not so for most real-world domains. The reason is two fold: Since modeling action outcomes precisely can be very difficult and many state variables are real valued, it is often the case that expectations and observations only differ slightly and in particular are qualitatively the same. Secondly, we often face a lot of unpredictable exogenous events most of which are unrelated to the problem the agent is facing and thus do not matter. But since still some of these may actually matter in certain situations, we cannot leave these details out of consideration entirely by pruning them from the model. We find that this aspect of execution monitoring has received very little attention in the past and in particular has never been approached formally – an omission we intend to address in our research. Section 4.3.2 describes the relevant existing work on this issue.

Replanning: The problem of replanning is that of modifying the plan or planning from scratch when the current plan has been asserted to be sub-optimal or invalid. The questions involved here are whether to try the one thing or the other, and, in either case, how this process can be performed quickly and what the constraints are. Classically systems aim at repairing a failed plan quickly, ignoring the quality of the plan and following the intuition that plan-repair is faster than planning from scratch. Section 5 addresses these issues.

We intend to focus our research on the last three items above.

2.1 Comparison To Existing Frameworks

Here we briefly compare our proposed framework with other recent proposals towards formalizing the problem of execution monitoring.

The presented framework is similar and indeed inspired by that of [Bjäreland, 2001]. Our Model entity replaces Bjäreland's "Expectation Assessment" function and our Observations replaces his "Situation Assessment" function. Further we merge his "Discrepancy Detection" and "Discrepancy Classification" steps into one Diagnosis step. Bjäreland's approach is driven by the notion of a discrepancy between what was expected and what was observed. However, we believe that it is not always necessary to determine the exact discrepancy and also that often a significant amount of state estimation has to take place to determine it precisely. Our approach seems in particular in probabilistic settings more appropriate where a discrepancy may not be well defined. As a major difference, Bjäreland's framework lacks the State Evaluation step, a step we consider crucial for the overall success and performance of the system. Finally, we name the last step Replanning rather than "Recovery", as not all discrepancies that matter are necessarily malignant and need to be recovered from but can also be serendipitous.

In [De Giacomo et al., 1998] the authors propose a framework formalized in the situation calculus for monitoring the execution of high-level robot programs specified in the GOLOG agent programming language [Levesque et al., 1997]. In this framework the authors assume to observe exogenous actions themselves, hence not addressing the problem of diagnosis. The monitor is defined through a predicate $Monitor(\delta, s, \delta', s')$ and 'called' after every step of the program. $Monitor$ exchanges the situation term s by s' to reflect the potential occurrence of an exogenous action, and modifies the program δ as appears adequate: If another predicate $Relevant(\delta, s, s')$ holds, a recovery predicate is called which may modify the remaining program δ to recover from the discrepancy, otherwise the program remains unchanged. While the focus of [De Giacomo et al., 1998] is on the formal specification of the framework they do propose a specific, but simple, monitor. The *Relevant* predicate, fulfilling the state evaluation step of our framework, is realized by simulating the remaining program δ in the new situation s'. If this simulation succeeds, that is the program can successfully be executed in the new situation, the discrepancy is irrelevant, otherwise it is relevant. Replanning is defined as finding a shortest repair program p (sometimes called a *patch* (cf. [Eiter et al., 2004])) for δ, such that the sequence of the two programs, $p; \delta$, is expected to successfully execute as verified by simulation (forward projection). This suggests that all discrepancies are considered malignant.

Finally, [Fichtner et al., 2003] formalized a similar framework based on the fluent calculus and implemented it on a robot using the fluent calculus implementation FLUX.

Our research interest and the focus of this document is however not on frameworks for formalizing but on methods for solving the problem of execution monitoring. We thus do not consider the advantages and limitations of these frameworks in detail.

We begin the technical part of our presentation by reviewing some early approaches to integrated planning and execution. In Section 4 we will review approaches to monitoring (see dashed box in Figure 1) and in particular in Sections 4.3.1 and 4.3.2 present relevant work for the Diagnosis and the State Evaluation steps, respectively. In Section 5 we discuss the existing approaches to replanning, elaborating briefly on approaches that refine the model prior to replanning in order to accommodate for the mis-prediction that caused the discrepancy.

3 Integrated Planning and Execution

In this section we review a few, in particular early, approaches towards integrating planning and execution. We can roughly divide the presented approaches into two classes, those that actively reason about discrepancies between what was expected during planning and what actually happened during plan execution, and those that do not.

3.1 Discrepancy-Based Approaches

Work on execution monitoring reaches back to the earliest deployed systems. As one of the first [Fikes et al., 1972] described, among other things, the execution system applied on Shakey the Robot. Even though the language used for planning, STRIPS, is very limited in expressiveness, the authors presented some interesting ideas for monitoring the execution of plans and reacting upon execution failure. The core idea is to represent plans, which here are limited to be sequential, in a way that supports monitoring by revealing their structure. For this, the authors introduced *triangle tables.* Figure 2 shows a triangle table for the sequential plan OP_1, OP_2, OP_3, where OP_i is a STRIPS operator. Each cell represents a set

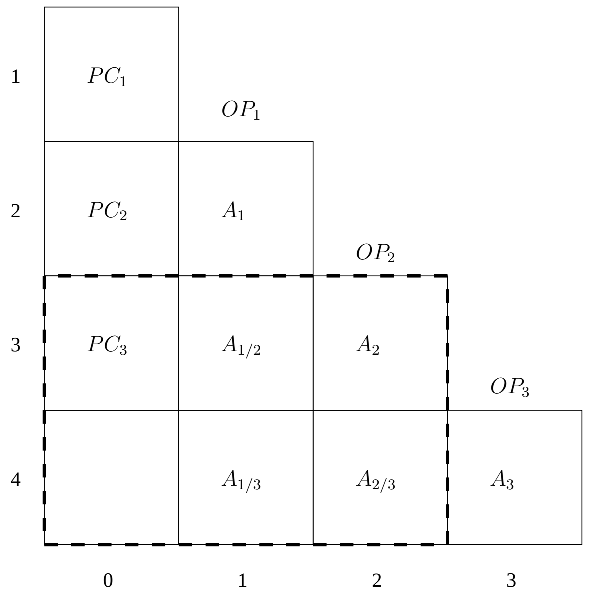

Figure 2: A triangle table for representing a plan with three operators $OP_1, \ldots, OP_3$, the dashed box defines the "3rd kernel".

of clauses. In the cells below each operator OP_i, those ADD effects of OP_i are contained that persist after the execution of a subsequent execution step. That is, A_1 contains exactly those clauses that are produced by OP_1 and $A_{1/2}$ contains that subset of A_1 whose clauses are not deleted by OP_2. The left-most column (number 0) holds those clauses that are preconditions to the corresponding operators in the plan but are not produced by earlier operators in the plan and are thus required to be established by other means. Taken together, these clauses (PC_i) define the overall preconditions for the entire plan. Further, Fikes et al. defined the *marked clauses* of a cell to be those clauses that are required by the operator of that row. Then the i^{th}*-kernel* is defined as the set of marked clauses in the rectangle that includes the bottom left cell and row i (see the figure for an example). The interpretation of this is that whenever the clauses of the i^{th}-kernel are true in a given model of the state and dynamics of the environment, then the i^{th}-tail of the plan, starting with operator i, is applicable and projected to reach the goal. In our terminology we say the remaining plan is *valid*. Further, if the $(n+1)^{th}$-kernel is true, where n is the number of operators in the plan, the goal is already achieved. Based on the triangle table representation of a plan, the execution strategy that Fikes et al. proposed, called PLANEX, proceeds as follows:

1. Test the highest kernel $(n+1)$.

2. If it is true, the goal is achieved, stop execution.

3. Otherwise, proceed checking lower kernels until one is found that is true and then execute the corresponding operator.

4. If no kernel in the triangle table is true, the plan has failed and replanning is required.

5. Otherwise, repeat.

By optimistically checking all higher kernels before executing the next operator in the plan, this strategy has the advantage that it discovers some serendipities. It also repeats parts of a plan that have failed to achieve their intended effects. However, this procedure may end up in an infinite loop when the failure is due to conceptual, e.g. modeling, errors like for instance assuming that an action achieves a certain literal but simply doesn't, no matter how often the action is executed.

While the applied action description language, STRIPS, is fairly limited, we make the following observation that allows us to lift the benefits of this approach to more expressive action description languages: Although the authors seem to have been unaware of this at the time of this article or they simply miss pointing it out, a kernel is exactly the regression of the goal through the remaining operators of the plan and the simplicity of the representation, merely based on clauses of the ADD effects, is due to the limitations of STRIPS to non-negative, non-disjunctive preconditions and goals, and the required absence of conditional effects of actions. This regression is produced by the applied backward-chaining planning algorithm and thus comes without computational overhead.

The utility of the plan structure for execution monitoring was later recognized by others. In these approaches the term *rationale* was coined, as the authors realized that not so much the structure of a plan but rather the rationale for the choice of its elements is what has to be preserved. This was the case with the hierarchical SIPE planning system [Wilkins, 1988] and in the IPEM system [Ambros-Ingerson and Steel, 1988] based on the TWEAK partial order planner. The execution monitoring strategy of the former is described in [Wilkins, 1985]. The core idea is to represent the purpose of every action in the plan by stating the time until when its effects have to be maintained, e.g. until the goal is reached or a depending action has been executed, and to explicitly mark assumptions made during planning. These annotations are used during execution to decide whether a discrepancy is affecting the remaining plan or not. When that is so, SIPE offers a set of eight replanning rules that 'often retain much of the original plan'. These rules are incomplete in the sense that they do not necessarily produce a legal plan but rather a task network with some unachieved sub-goals remaining, so the transformed plan is handed back to the planner.

In many applications, it is desirable to conservatively, i.e. only minimally, modify the current plan when needed, as this is generally assumed to minimize execution costs. An example of this is described in [Myers, 1998] where the SIPE approach was used to monitor the execution of air campaigns. We, however, believe that if there are costs involved, these should be modeled explicitly and the monitoring be extended appropriately to monitor continued optimality of the plan according to these costs as opposed to only monitoring continued validity and soft-constraining the replanning implicitly through an unspecified preference criterion as Myers does.

Similarly to SIPE, Ambros-Ingerson and Steel approached execution monitoring in conjunction with the TWEAK partial order planner [Ambros-Ingerson and Steel, 1988]. They extend the set of plan transformation rules of the planner to accommodate for unforeseen events and to integrate the execution into the process. The actual execution monitoring is then performed through the application of predefined IF-THEN rules mapping *flaws* to *fixes* (plan transformation rules). A scheduler heuristically sorts the list of open flaws continuously during execution. While the paper does not go into detail about how to efficiently detect flaws, which is a major limitation, this work is still interesting as it shows how partial-order planning can naturally perform plan repair upon unforeseen events during execution. This is due to the fact that partial-order planning, much like backward-chaining, chooses actions to add to the plan based on open preconditions and this again makes the reasons for adding the actions, the rationale, explicit and usable for monitoring (and plan repair).

While PLANEX merely used the plan's rationale for evaluating whether the current plan or parts of it are still valid for achieving the goal, SIPE and IPEM also exploit this information for replanning.

Others have approached the problem of recovering from discrepancies during execution heuristically using predefined fault models and fixes. Beetz and McDermott [Beetz and McDermott, 1994] described how XFRM, a planner based on RPL, the Reactive Plan Language, handles discrepancies. In this framework, the model of the dynamics is probabilistic, i.e. actions can have stochastic effects. When a discrepancy arises during the execution of a plan, sample execution scenarios are projected to evaluate the impact of the discrepancy. This contrasts with the earlier presented approaches which, one way or the other, all implicitly regressed the goal to evaluate the impact of the discrepancy. Here instead the discrepancy is progressed. Since the projection is non-deterministic, the number of sample scenarios determines the probability of detecting a problem, but the authors do not investigate how many samples are required depending on the situation and the domain, neither do they provide a concise definition of a failure. When failures are projected to happen, these are classified into a domain dependent hand-crafted taxonomy of fault models that also maps them to plan transformation rules. The application of these transformations may improve the plan, but is not guaranteed to do so. All in all the approach is incomplete in at least two senses: due to the choice of samples certain problems arising from a current discrepancy in the future may not be detected and due to the heuristic character of both diagnosis and replanning (plan transformation), the approach is not guaranteed to recover from contingencies.

In [Beetz and McDermott, 1996] the authors later extended the approach to improve flexibility between planning and execution, but this didn't address any of above criticism.

3.2 Approaches Not Based On Discrepancies

Reactivity was the motivation for the latter heuristic approach, that is, the desire or requirement to respond quickly to execution failures. Reactivity was also the driving force for so called *universal plans*, proposed by [Schoppers, 1987]. The idea is simple: starting from the goal, backward-chaining is performed until either a contradiction is produced or no open sub-goal (precondition of some action in the tree) has support, i.e. there is no action whose effects satisfy any of these open sub-goals. This creates a decision tree that dictates which action to perform in which state in order to reach the goal. The approach is indeed universal in the sense that for any state for which there exists a plan of getting to the goal, this plan can readily be read off the tree. This way, whatever goes wrong during execution of a plan, the system instantly knows how to react if it is possible to reach the goal from the new state.

Unfortunately, the price for this is high and in fact makes the whole approach intractable: in worst case the time and space complexity is linear in the size of the domain, which, if not infinite, is exponential in the number of domain features. Further, this was just a reinvention of *policies* as defined in decision-theoretic planning in MDPs (see for instance [Puterman, 1994]), with a factored state space. This is comparable to recent approaches to first-order decision-theoretic planning using regression ([Boutilier et al., 2001]).

As with universal plans, in decision-theoretic planning generally no knowledge about the initial state is assumed but the problem is solved in a backward fashion for all states. This is generally infeasible for the stated reasons (also cf. Section 6.1). When the initial state is known, i.e. when a policy is not computed ahead of time but when a particular problem instance arises, forward search can be use. For that matter a forward search based approach to decision-theoretic planning was presented by [Dearden and Boutilier, 1994]. This approach performs conditional planning for the outcomes deemed possible for the considered actions in the current situation, but prunes parts of this search tree where it is sound. The approach relies on the existence of a heuristic function and Boutilier et al. also present ideas on how such a function could be constructed. Pruning is possible when upper and/or lower bounds on this function are known and when the error of this function, compared to the actual value of a node, can be bound. The general assumption in execution monitoring is that enumeration of all contingencies is impossible. But even when this is not the case, i.e. certain abnormalities can be predicted, it may not be wise to try to conditionally plan for all of them as some may be very unlikely and their consideration generally blows up planning time exponentially. Instead, one may want to investigate the possibility of recovering from these contingencies only when they arise. The work by [Dearden and Boutilier, 1994] may be a first step towards investigating how much conditional planning makes sense and when it is safe not to plan for a potential outcome. The decision of leaving anticipated but unlikely abnormalities out of consideration in planning could be a result of meta-reasoning (cf. Section 6.2).

Probably the most prominent recent applied work in execution monitoring is that of [Williams and Nayak, 1996], where the authors describe the Livingstone system that flew on NASA's Deep Space One mission which successfully tested high-risk technologies in space and collected scientific data and images of a comet from 1998 to 2001. Livingstone is a model-based reactive self-configuring system. It uses propositional logic augmented by a 'next' operator to model a transition system, where all occurring propositions are equalities $y = e$, with y a state variable and e an element in the domain of y. Based on that, the model-based configuration manager determines control values for a set of control variables in two steps: in the mode-identification step (MI) the set of plant trajectories that are consistent with previous control values, sensor readings, and the transition system is generated. In our framework this would be called the monitoring part. From that, the mode-reconfiguration (MR) generates sets of control values that immediately lead to a goal state, characterized as a subset of states (configurations). This would be the replanning. The system only considers one-step plans (reconfigurations) and makes strong assumptions about the countability of states and transitions (actions). In general we cannot assume that the system can recover from a discrepancy in just one step, or that a goal can be reached after a single transition. On the other hand, this work makes a limitation of all above works apparent: not only the current state may differ from what was expected but also the *goal* may – and does – change in realistic environments. The methodology of this approach more closely resembles that of control theory (cf. Section 6.3) rather than AI planning.

4 Monitoring

Monitoring, in the context of execution monitoring, is the problem of deciding whether execution of the current plan should be continued or not. In this section we review work that is relevant for this task and for the sake of presentation we roughly classify the presented approaches into one of three categories:

1. *external monitoring*, where the monitoring and the executing systems are disjoint,

2. *expectation-based monitoring*, where the decision is based on the materialization of defined expectations, and

3. *model-based monitoring*, which is the approach we assume in our framework (cf. Figure 1), that is, sub-dividing the task into diagnosis and state evaluation.

In the latter approach a lot of work can be found on the diagnosis part of the problem (Section 4.3.1), while the evaluation problem has attained very little and in particular not much principled work (Section 4.3.2).

4.1 External Monitoring

In what we here call external monitoring, the monitoring and the executing entity are independent, meaning that the monitor watches the agents behavior but does not have access to the agents internal state. In particular the monitor is not informed of the actions being executed but can only try to infer this information from observing the actions' effects. Although this was proposed in the literature to enable one agent to monitor other agents, this work may still be of interest even in frameworks like ours where we do not separate these two entities, as this approach provides insight into the difficult diagnostic problem of deciding 'what happened', a question we are going to return to in Section 4.3.1. But in contrast to the approaches in Section 4.3.1, the approaches presented here also evaluate the found diagnoses.

The Autominder system ([Pollack et al., 2003]) is a cognitive orthodic helping people with memory impairment by issuing personalized reminders to help the client accomplish her daily agenda. The system observes the client's actions to infer whether a reminder is necessary or not. The authors of the system argue that since sensory input is noisy and the action in the client's plan (agenda) can have complex temporal constraints, reasoning under uncertainty and reasoning with quantitative temporal relationships between events has to be integrated for this task. For this purpose they introduce Quantitative Temporal Dynamic Bayesian Networks (QTDBNs; [Colbry et al., 2002]), an integration of Time Nets and Dynamic Bayesian Networks (DBN). The Time Net component models the relationship of the time intervals client actions occur in, representing the probability that an event occurs in a certain time interval, while the DBN reasons about actions, sensors, and domain properties within any such time interval. The DBN is updated with the arrival of new information, i.e. sensory input, while the Time Net is updated only as the actual time crosses a time interval boundary. At these time changes two interface functions pass information from the DBN to the Time Net and back. Experimental results show that for sufficiently small tasks, QTDBNs achieve their purpose of monitoring the execution of another agent's, here the client's, plan, but the required Bayesian reasoning, which is known to be NP-hard, causes a time complexity exponential in the number of modeled actions.

To monitor the execution of a multi-agent system, [Dix et al., 2003] investigate an approach that could be described as meta-planning. Again unaware of the internal states of the participating agents in the system, the authors propose to create a set of *intended plans* off-line in a meta-theory that reasons about messages passed between the individual agents partially revealing the executed actions. On-line, these intended plans are filtered based on the messages actually passed between agents. That is, when a new message is observed, the set of intended plans is reduced to those plans compatible with this observation. The system raises an alert to the user if the set becomes empty, to indicate that according to the meta-theory there is no way the monitored system is going to achieve its objectives, or when no message has arrived for a specified time, indicating that the system is stuck. Hence, instead of proving that the monitored system performed nominally, the approach here is to try to prove the opposite and assume nominal operation as long as there is at least one possible way of reaching the goal, i.e. at least one compatible intended plan. This approach demonstrates that monitoring is possible even when the applied models in the execution system and the monitoring system are different, which contrasts to the approach in Autominder, where the client's plan is known precisely and taken into account for monitoring. Instead, the monitoring system here merely bases its decisions on the objectives of the agents being monitored and a rough indication on which actions are being performed, perceivable from the passed messages. It also allows monitored agents to change their plans, as long as there remains a way of reaching the goal. Hence the goal and not the plan is used to determine failure.

Even less knowledge about the monitored system is required when learning techniques are applied. To decide whether the execution of a task will still finish before a deadline is reached, [Hansen and Cohen, 1992] model the problem as a sequential decision problem. This can either be solved by standard techniques when a model of the actions and their impact on reaching the goal is known, or using temporal difference methods otherwise.

Although interesting from a conceptional point of view, for us this is not a viable options as we are concerned with monitoring the execution of a plan correctly right from the first time it is executed and cannot afford to perform test runs first to collect learning data.

As we pointed out earlier the techniques of this section are not promising approaches for us per se, as they abstract too much from the underlying system and disregard information we gage valuable in both monitoring and replanning.

4.2 Expectation-Based Monitoring

The idea of basing the monitoring decision on the materialization of certain plan-dependent expectations was first stated in [Doyle et al., 1986]. Doyle et al. proposed an approach to the problem of monitoring the successful execution of a plan through the generation and use of perception requests to verify the nominal execution of the involved actions. Given a plan and a model of the environment, their method selects properties that need to be monitored and generates perception requests that can verify the materialization of these properties. The latter are embedded into the plan. The properties are typically pre- and post-conditions of actions, and perception requests are sensing actions together with the expected sensing values. During execution the actual sensing values returned by the sensing actions can be compared to the expected values to detect discrepancies. While the significance of the actual implemented system described in the paper called GRIPE (Generator of Requests Involving Perception and Expectations) may be questionable, the paper is notable for raising the awareness of relevant questions. These involve how often and accurately assertions should be verified and how this can be done best, i.e. introducing the problem of planning for perception requests. But first and foremost the question as to which properties should be monitored is raised and this question today is still awaiting a satisfactory answer. The authors propose four criteria involved with this choice:

- the *uncertainty* of properties due to several reasons (incomplete initial knowledge, stochastic action outcomes, etc.);
- the *dependency* of future actions on the property;
- the *importance* of the property; and
- the *ease of recovery*.

Of these four the second has received the most attention as we have seen implicitly already in PLANEX, SIPE and IPEM, and repeat to see in the sequel of this survey. Doyle et al. realized that some action effects are irrelevant as no future action nor the goal require their presence, while others, said to lie on the *critical path*, are. But the paper lacks a formal definition of this set. The notion of a critical path relates this work to the triangle tables we saw earlier since 'elements of the critical path' is merely a synonym for 'marked clauses in a kernel'. Doyle et al., like Fikes and his colleagues earlier, did not realize that this is again just the regression of the goal through the remaining actions of the plan. Characterizing this technique as goal regression, or more generally rationale regression, will likely enable the generalization of this technique to other action formalisms, planning paradigms, and preference criteria.

Inspired by this work, Musliner et al. discuss some aspects of time in execution monitoring for actions with durations and in particular address the questions of how to verify critical assertions most effectively ([Musliner et al., 1991]). By using a planner that deploys simple depth-first backward-chaining from the goal to the initial state, they construct the critical path. Then, at every stage in the plan where a post-condition is established that is used in the precondition of a later action, a verification action is inserted to monitor the condition over the required time interval. Musliner et al. argue that these actions themselves may require planning as they too have pre- and post-conditions and durations. To accommodate that fact, they propose to augment the theory with models for these sensing actions. The paper however lacks formal details and leaves certain questions open like, e.g., whether and how the models for the normal actions are modified to motivate the introduction of verification actions.

Another examples of expectation-based monitoring is the work by [Earl and Firby, 1997]. Instead of planning from first principles, the presented Routine Management and Analysis system (RAMA) chooses from predefined *dynamics* when given a goal to achieve. These dynamics combine actions and expectations about observations to be made during their execution.

Expectations were also the basis for the method we recently proposed for monitoring the execution of plans generated from GOLOG programs, where the produced plan is explicitly marked with the assumptions that were made during planning [Ferrein et al., 2004]. These assumptions represent the at planning time projected truth values of conditions included in the GOLOG program (e.g. in if-then-else constructs or while-loops). If during execution of the plan these assumptions are violated or the next action is not possible, the plan is abandoned and replanning from scratch is performed. This seems to be the most reasonable thing to do in a domain that is as dynamic as RoboCup where we applied this work.[1] This contrasts with the earlier mentioned work of [De Giacomo et al., 1998] as it does not monitor the execution of the GOLOG program itself, but the resulting (conditional) plan after performing decision-theoretic planning over a program, maintaining the expectations involved through the hard constraints in the original program. While the approach worked in practice, it cannot claim any kind of correctness or optimality. For instance, it does not anticipate failure of future actions in the plan, like others do. Nevertheless it serves as a rare example of monitoring where the constraints underlying the construction of the plan, which in this case were procedural hard constraints represented through a GOLOG program, are taken into account in monitoring the execution of the plan. This is crucial, because when these constraints change, there is no reason to believe that the plan at hand continues to be optimal or even valid.

Consider the following "Tree Example" of [Lespérance et al., 2000], which we modify slightly for our purposes. In this domain our agent is trying to fell a tree and carry it home.

[1]We used the presented approach in several RoboCup tournaments including the world-cups of 2003 and 2004.

To not faint from exhaustion she has to rest when her fatigue is getting too bad. She thus can decide between three actions, `chop`, `rest`, and `carry-tree`. Her doctor told her to rest whenever she feels high fatigue. Overall we can incorporate these hard constraints with the original task into the following GOLOG program

$$(\textbf{if}\ \neg\text{fatigue}\ \textbf{then}\ (\texttt{chop} \mid \texttt{carry-tree})\ \textbf{else}\ \texttt{rest}\ \textbf{endIf})^* \ ;\ \text{tree-at-home?}$$

where $a|b$ denotes the nondeterministic choice between two sub-programs a and b, $a;b$ denotes their sequence, and a^* denotes the nondeterministic iteration of a. A tree-chopping expert said that it usually takes five chops to fell a tree and our agent has an idea of how much labor, chopping and carrying, she can do until she needs a rest. From this model information and the hard constraints a GOLOG interpreter may produce the following plan: `chop, chop, chop, rest, chop, chop, carry-tree`. However, when executing this plan it happens that our agent feels fatigue already after two chops, the model was imprecise. The remaining plan is still executable but would violate the given hard constraints and would cause our agent to faint. The problem is that the hard constraints got lost in planning because the planner assumed correctness of the model. To account for such modeling errors the method proposed in [Ferrein et al., 2004] would produce the following annotated plan instead: $\mathfrak{M}(\neg\text{fatigue})$, `chop`, $\mathfrak{M}(\neg\text{fatigue})$, `chop`, $\mathfrak{M}(\neg\text{fatigue})$, `chop`, $\mathfrak{M}(\text{fatigue})$, `rest`, $\mathfrak{M}(\neg\text{fatigue})$, `chop`, $\mathfrak{M}(\neg\text{fatigue})$, `chop`, $\mathfrak{M}(\neg\text{fatigue})$, `carry-tree` and the proposed execution engine discards the plan if during execution a marker ($\mathfrak{M}$) is encountered whose condition fails to hold, contrary to what was expected.

While most work on monitoring is concerned with monitoring the execution of a plan, it is also possible that even during planning pieces of the tentative plan become invalid, for example when a precondition that was true earlier becomes false. This aspect of planning and monitoring in dynamic domains is addressed in [Veloso et al., 1998] where the resulting approach is implemented in the PRODIGY framework ([Veloso et al., 1995]). In particular the authors propose to generate two kinds of monitors during planning: plan-based monitors targeting preconditions of actions in the tentative plan, and alternative-based monitors. The latter are relevant when preferences over different possible plans exists: during planning it may happen that the most preferred alternative is not possible for some reason. If this reason changes and the alternative becomes available, optimality requirements force us to pursue this alternative instead, so monitoring this reason is vital for optimality. While this presents an interesting aspect, in the described approach the problem is strongly simplified by assuming that the quality of an alternative can be decided a-priori, that is, without exploring the corresponding part in the search tree first and by assuming that the quality of a plan is purely determined by its comprising actions and in particular independent of the traversed states. This is a major limitation of this approach, which also lacks a formal foundation.

4.3 Model-Based Monitoring

Recall our framework of Section 2, where we sub-divide Monitoring into Diagnosis and State Evaluation. Evaluating the state with respect to the plan, that is, determining whether the plan continues to be valid and optimal or not, may require more information than provided by the observations. Supplying this information is one of the tasks of diagnosis, estimating the state. When the state estimation leads us to think that the world has evolved differently from what was predicted by the model, diagnosis also serves the purpose of answering the questions "what went wrong?", "what did actually happen?". This is in particular useful to know in situation where the model needs to be refined before replanning, to avoid getting caught in an infinite loop (cf. Section 5.3). The two tasks are obviously related and can even be indistinguishable in certain state representation like for instance the situation calculus (e.g. [Reiter, 2001]).

4.3.1 Diagnosis

In this section we review selected previous work relevant to the problem of Diagnosis in the context of execution monitoring. We will first briefly review early work on static model-based diagnosis and then explain how this relates to the problem of state estimation and the diagnosis of dynamical systems.

Model-based diagnosis was first described by Ray Reiter ([Reiter, 1987]). In Reiter's approach the system description, SD, is a finite set of first-order sentences and in this system a set of components, $COMP = \{c_1, \ldots, c_n\}$, exists, each of which may either perform abnormally ($Ab(c_i)$) or nominally ($\neg Ab(c_i)$). Given SD and $COMP$, an observation OBS, represented as another finite set of first-order sentences, *conflicts* with the assumption that all components work correctly if $SD \cup \{\neg Ab(c_1), \ldots, \neg Ab(c_n)\} \cup OBS$ is inconsistent. The problem of diagnosis is then to find a subset Δ of the components such that assuming these components abnormal and the rest to work nominally, consistency of the union with SD and OBS is reestablished. The subset Δ is called a *diagnosis*. Generally there are several possible diagnoses in which case one is generally preferred over the others. In Reiter's approach this preference is defined through minimality, that is, a diagnosis Δ is preferred over another diagnosis Δ' if and only if $\Delta \subset \Delta'$. The preference for minimal diagnosis also makes sense from a probability point of view. Assuming that correct behavior of a component is more likely than its failure and that component failures happen independent of each other, minimal diagnoses are also most likely diagnoses.

[Witteveen et al., 2005] have used this approach to diagnose abnormal events during the execution of agent plans. The system description models the dynamics of the domain, in particular the effects of the agent's actions, and the components are the agent's actions themselves. In this setup the Independence of component failures is not given anymore, as the failure of one action can causes others to fail to. The authors accommodate for this fact by introducing the notion of *Pareto minimal causal diagnosis*. Roughly, the original diagnoses are reduced to their causes when a subset of components in the diagnosis causes the failure of other components in the set. Minimality is then determined based in the reduced sets.

Reiter later extended his original work with de Kleer and Mackworth ([de Kleer et al., 1992]) to allow for fault models and exoneration axioms, which violate the assumption implicitly made in the original work that also every super-set of a diagnosis is a diagnosis itself. The main approach presented in this paper was based on the notion of prime implicants and the diagnoses defined from that were called *kernel diagnoses*.

Another way of defining the most probable diagnosis is the introduction of explicit numeric failure probabilities. The generation of candidate diagnoses can then be focused to the most likely ones ([de Kleer, 1991]). To improve the efficiency of diagnosis, it is often also beneficial to compile the diagnostic model into a representation which is more efficient for the diagnostic task, but this often increases the space required by the representation. Provan ([Provan, 2005]) recently addressed this problem by again exploiting the preference information (probabilities) and limiting the parts that are compiled based on that.

The task of state estimation is arguably similar in nature to the above described problem of diagnosis: given some observation that gives rise to the suspicion that the actual current state is not the one we expected, we would like to identify the actual state as best we can. As such the above described diagnosis is a special case of state estimation where the incompleteness of knowledge is limited to the abnormality of the components. It is also limited as it does not provide for dynamics in the system, that is, it is only possible to talk about what holds or doesn't hold *now* but not what may have happened in the past to reach the current state.

McIlraith addressed this misfit in [McIlraith, 1997]. She combined above work with the situation calculus to model action dynamics and introduced the notion of explanatory diagnosis. Given a basic action theory in the situation calculus Σ, a history of actions $HIST$ that is known to have happened since the initial state S_0, and an observation OBS, an *explanatory diagnosis* is a sequence of actions $E = \alpha_1; \ldots; \alpha_n$ such that $\Sigma \models Poss(HIST; E, S_0) \wedge OBS(do(E, do(HIST, S_0)))$. That is, an explanatory diagnosis is a sequence of actions that may have happened following $HIST$ and explains the observations. These actions are assumed to be *exogenous*, that is, not under the control of the monitored agent. The author shows that the problem of finding an explanatory diagnosis coincides with the problem of planning: The system dynamics are described by the action theory Σ, the initial state is the state resulting from executing $HIST$ in the initial state S_0, $do(HIST, S_0)$, and the goal is described by the observations OBS.

McIlraith's work was extended by [Iwan, 2000, Iwan and Lakemeyer, 2003]. Iwan realized that in order to explain the observations, it is sometimes not enough to conjecture the occurrence of exogenous actions after the given history of actions $HIST$ but that in order to explain the observations one has to assume the occurrence of exogenous actions in between the given action history and/or that some actions in this sequence were not performed as expected. Iwan also addressed the problem of characterizing and computing the most preferred diagnosis using explicit probabilities for the occurrence of events, much like [de Kleer, 1991] proposed for the static case. The computation is based on best-first forward search.

Both McIlraith and Iwan make the assumption that observations are only made at the end of the action sequence to be conjectured or equivalently that diagnosis is performed right when the observations are made. This may prove to be too strict a limitation in practice as in real-world systems computational resources are limited and thus it may not always be possible to perform the diagnosis right away when a surprising observation is made.[2] Also,

[2]Without further computation, it is also not always possible to say whether an observation is surprising

due to incomplete knowledge, it may be the case that certain observations do not contradict the model assumptions immediately but only after additional observations are made or simply have a delay. This suggests to extend the presented work by using approaches of planning with temporal goals, in order to describe the temporal relationship between observations and presumably performed actions. Recent work by Baier et al. ([Baier and McIlraith, 2006]) can supposedly be lifted for this purpose. Also, to express more complex probability criteria over several possible diagnoses, work by Bienvenu et al. ([Bienvenu et al., 2006]) may help in defining and planning with qualitative temporal probabilities. The use of qualitative probabilities also avoids the problematic elicitation of numeric probabilities from experts, which may be difficult, annoying for the user, and error prone.

Another shortcoming of these approaches is their applicability to real-world systems involving continuous evolutions of real valued features, like for instance the three-dimensional position coordinates of a helicopter in operation. Consequently, the majority of deployed robotic systems described in the literature approach the problem of state estimation differently. The main difference is the representation of the dynamics: instead of sets of logical sentences for representing states and some kind of effect axioms for describing the effects of actions in the domain, these systems describe the state by the numeric values of certain properties and control values, and use algebraic or differential equations to describe the impacts these properties have on each other depending on the current mode of operation. This way not only continuous values and continuous time becomes manageable, but also continuous probability distributions can be modeled to capture the uncertainty of the domain.

This approach was used in [McIlraith, 2000] and [de Freitas et al., 2004]. In both these papers the system to be diagnosed was modeled as a hybrid system, that is, using a state representation that has both a discrete and a continuous part. The continuous part generally describes the dynamics, whereas the discrete part describes the operational modes. The latter induce different dynamics in the continuous part, for instance the direction of travel of a robot. Faults are defined through fault modes in the discrete part and cause the dynamics of the continuous part to change. Observations, on the other hand are generally only made in the continuous part, except for deliberate control mode changes. The task then is generally to infer if and when a faulty mode has materialized and reasoning is generally based on the belief state, that is, a probabilistic distribution over possible system states.

In [McIlraith, 2000], McIlraith casts the problem of diagnosing a hybrid system as a Bayesian model tracking and selection problem. To efficiently track multiple models simultaneously, she proposes the use of particle filters. One major problem with the use of particle filters for diagnosis is that they focus on the most likely models, that is the nominal behavior, while fault modes are unlikely and can therefore slip the attention of the, approximate, filter. McIlraith overcomes this problem by biasing the samples towards the results of a separate, qualitative diagnosis as it is described in [McIlraith et al., 2000]. The paper makes a single-fault assumption.

Particle filters now are a very common approach to approximate the distribution of the state variables. These can also be integrated with exact methods like for instance Kalman

or not.

filters into so called Rao-Blackwellised particle filter (see e.g. [de Freitas et al., 2004]). [Verma et al., 2002] combine particle filters with Partially Observable Markov Decision Processes (POMDP) for controlling a system: a policy for the POMDP is computed off-line while particle filters are used on-line to track the belief state. Particle filters can also be used for approximate inference in Dynamic Bayesian Networks (see e.g. [Russell and Norvig, 2003], pp. 565–568).

Depending on the way the system is modeled and on the available observations, diagnosis may not be required or can be trivial. This is for instance the case when the state can be sensed completely, or when it can be sensed partially and the observations coincide exactly with the predictions of the model. Since then there is no discrepancy, there is no reason to believe that the actual state is any different from the predicted one. In any case, the situation-dependent need and requirements for diagnosis should be guided by its purpose, that is, it should be determined with respect to the following steps, State Evaluation and Replanning. If, for instance, State Evaluation is able to specify a sub-set of states in all of which the current plan should be continued, then there is no need, and really no point, in trying to disambiguate between two candidate diagnoses when both candidates belong to this sub-set.

4.3.2 State Evaluation

Recall that in our framework (cf. Figure 1) the diagnosis step is followed by a state evaluation step to determine the relevance of the discovered and diagnosed discrepancy between the prediction and the estimated state, to decide whether any kind of replanning is required or advisable. This question has received little principled attention in the literature.

One can generalize the technique implicitly used by several approaches to answer this question, as that of annotating the plan at every step with the regression of the goal through the remainder of the plan (cf. Section 3). When the regressed goal holds in the state actually encountered during execution, the remainder of the plan is expected to succeed, according to the model. PLANEX uses STRIPS as the action description language and since STRIPS allows for neither conditional effects, nor disjunctive preconditions, nor uncertain action outcomes (disjunctive action effects), regression becomes very simple. This regression is implicitly done in the backward-chaining search performed by the planner and the annotation is part of the triangle tables used for representing plans. Our generalization also applies to SIPE, which is based on hierarchical planning, and IPEM, based on a partial-order planner. In both approaches the choice of actions to add to the plan is based on open preconditions (sub-goals), starting from the goal, and in both approaches the dependencies between these sub-goals and the actions in the plan they are established by are represented in the plan. Again these representations are used during execution to verify the continued validity of the remaining plan and to support replanning at a basic level.

Neither of the recent logic-based frameworks for execution monitoring [De Giacomo et al., 1998, Fichtner et al., 2003, Bjäreland, 2001] mentions or formalizes this matter. Both [Fichtner et al., 2003] and [Bjäreland, 2001] lack an evaluation step entirely.

The specific monitor suggested in [De Giacomo et al., 1998] performs evaluation not by regression at planning time, but through projection at run-time. This is done in the *Relevant* predicate (cf. Section 2). This is also the method used in [Soutchanski, 2003b]. The advantage of the regression based approaches is that the regression and plan annotation can be done off-line during or after planning, minimizing the time required on-line. The disadvantage is that it requires more memory to store the plan annotations.

Further, virtually no work exists on extending this approach to gaging relevance of an execution-time discrepancy to plan optimality. Since almost all deployed systems are pursuing optimality according to some quality measure, this appears to be a relevant problem to address in future research.

Recently [Boutilier, 2000] described a decision-theoretic model of monitoring the preconditions of actions in a plan during execution to determine whether or not the current plan should be continued, and proposed heuristic methods to make this otherwise intractable problem tractable for more than just very short plans. His motivation was three drawbacks with existing approaches[3]: (i) they generally ignore the monitoring cost, (ii) they do not account for monitoring errors (noisy sensors), (iii) they ignore the fact that a failed precondition now may be reestablished at the time when the affected action is to be executed (e.g. hearing about a traffic jam on a route that won't be reached for several hours). To address these concerns Boutilier modeled the decision of continuing or abandoning the plan as a POMDP. The state space of this POMDP is the set of vectors of truth values for all preconditions in the plan. At every stage of plan execution the POMDP can choose for each precondition in the remainder of the plan whether to monitor it or not, and after monitoring is done, whether to continue executing the current plan or to abandon it in favor of adopting the best alternative at this point. This is similar in methodology to the approach by [Hansen and Cohen, 1992], but for the general question of whether or not a plan should be continued, and not restricted to the question whether a deadline will be reached or not.

The approach requires the availability of certain information, part of which may be difficult to obtain in practice:

- For any point in the plan the value of the best alternative plan at that point has to be known, since it will be used as the value for abandoning the current plan. This is not generally known as common planners do not provide this information since that would incur greater planning time costs. Many, in particular currently popular heuristic search based planners, would provide an upper bound on this value however.

- The probability of possible failures has to be known. Although these probabilities certainly exist, estimating them may be difficult. This assumption also implies that the agent is aware of all possible faults and this again is not generally the case in the real world.

- A monitoring (e.g. sensor) model has to provide the likelihood for a particular sensor reading for the case that a particular precondition has failed, and for the case that it has not failed. This is to allow for monitoring errors (e.g. noisy sensors).

[3]including approaches that replan whenever an unexpected state is reached

The general POMDP defined this way is too complex to be tractably solvable. Instead Boutilier proposed decomposition and approximation techniques. These make the plan monitoring problem solvable in a reasonable amount of time, even for long plans involving hundreds of steps, while compromising only little on quality as shown by experiments.

Boutilier assumed away many complicating factors to keep the presentation simple. It is clear that the approach generalizes to cases where these assumptions do not hold, but it is unclear how that would affect the complexity. For instance are all preconditions assumed to be established prior to plan execution, i.e. the system does not have to reason about which actions are establishing preconditions of later ones and adjust the monitoring decisions accordingly. Also are conditional effects left out of the picture this way, but it is generally not enough to just monitor the preconditions of actions, but also the conditions under which certain desired or required effects are produced are relevant. It could hence be worthwhile trying to combine this approach with a rationale-based approaches, which can help in discovering the structure of a problem. Another limitation is the fact that alternative plans are not monitored at all. This is problematic because if during execution an alternative becomes better than the current plan, we should adopt it. Also if the best alternative decreases in value, this should affect our decisions as it may no longer be advisable to adopt the current alternative plan just yet when some future precondition of the current plan is expected to fail. Overall, Boutilier addresses the question "whether" and "when" to monitor relevant conditions, but he does not cover the question "which" conditions are relevant. By assumption the set of relevant conditions is already given but finding this set is not trivial.

5 Replanning

Once we have estimated the actual state of the world, evaluated it with respect to the plan, and asserted that the current plan has become invalid or sub-optimal, we have to decide how to react to this. In particular we do not want to replan from scratch in the new state but should try to repair our current plan accordingly. Or not? Nebel and Koehler [Nebel and Koehler, 1995] showed, in what is about the only theoretical paper on this topic, that modifying a given plan for an altered initial and goal state has the same complexity as planning from scratch. This holds even when similarity between the planning tasks is assumed and as little as one atom[4] is removed from or added to the goal while the initial state remains unchanged. The result also holds for the reverse case of a minimally altered initial state, the common situation in execution monitoring after a discrepancy occurred. When *conservative*, that is minimal, modification of the existing plan is required, the situation becomes even worse. Then plan modification can be even more complex than planning from scratch, that is, there are cases where planning from scratch can be done in time polynomial in the size of the problem definition while minimally modifying an existing plan is NP-complete.

While this suggests to not even bother trying to repair a failed plan but to simply replan from scratch, several people have found plan modification to be more efficient in practice

[4]Nebel and Koehler base their considerations on propositional STRIPS planning.

[Kambhampati, 1990, Gerevini and Serina, 2000, Koenig et al., 2002, Hanks and Weld, 1995] as we will see in detail in Section 5.1. The worst case complexity results should thus be taken with a grain of salt and in particular it would be interesting to further investigate the conditions under which plan modification becomes less efficient. The proposed similarity measure based on the removal or addition of unspecified atoms from or to the goal or initial state seems not very informative in this respect.

Another reason why Nebel and Koehler's results may not be very relevant to us, is the fact that they limit their considerations to STRIPS planning without preferences over plans. It remains to investigate whether these results also apply to plan modification under planning with preferences. This raises a much broader issue regarding the available literature on execution monitoring and replanning in particular: In almost all approaches the objective in replanning is to minimize the replanning effort, *not* to maximize the resulting plan's quality. This was also one of the concerns in a recent paper at ICAPS: Cushing and Kambhampati [Cushing and Kambhampati, 2005] argue that generally plan optimality is desired and planning time is only a secondary objective, but current replanning methods do not account for that. The naive solution for achieving optimality is of course continuous planning, i.e. replanning from scratch after every execution step. This strategy was for instance used in [Lazovik et al., 2003], where the authors present a framework for planning and monitoring the execution of web service requests.

[Cushing and Kambhampati, 2005] also raise concerns about the common limitation to replanning for altered initial and/or goal states. Some changes in the world, in particular those affecting the available operators, cannot be modeled under these assumptions. Imagine for instance a robot who breaks her gripper and cannot lift objects anymore. Cushing and Kambhampati propose to precede replanning with a model-adjustment step to alter the planning operators as necessary to accommodate for this kind of discrepancies. Not doing this, implicitly assumes that a failure is never the agents 'fault' and this ignorance can lead to the infinite repetitions of a (systematic) mistake. We will come back to this issue in Section 5.3.

5.1 Plan Repair

Despite the theoretical worst-case results by Nebel and Koehler, many people have shown plan modification more efficient than planning from scratch in practice. The motivation for this work has not always been execution monitoring, it was also explored as a means of more efficient planning, so-called case-based planning or planning from second principles. Instead of planning from first principles when a planning problem arises, the idea is to consult a library of preexisting plans, find one that matches well with the new problem, and then modify it according to the new requirements. The problem of replanning in execution monitoring is a special case of this where the plan library consists of only one plan, the current but failed plan. In our presentation we do not distinguish the presented results by their original motivation and for those approaches rooted in case-based planning, we ignore the methods for plan retrieval from a library.

Kambhampati describes a plan modification framework based on the PRIAR hierarchical task network (HTN) planner [Kambhampati, 1990]. This paper is of particular interest as it again demonstrates the utility of the rationale, now in particular for the plan modification/repair task. Again the plan is annotated with the rationale, the goal regressed through the remainder of the plan. Kambhampati calls this annotation the *validation structure* of the plan. As the name suggests, the validation structure serves to verify the plan's validity. A *validation* is a 4-tuple $\langle E, n_s, C, n_d \rangle$, where n_s and n_d are leaf nodes of the HTN, i.e. primitive actions, E is an effect of n_s (the source) and C is a precondition of n_d (the destination). Each node n in the HTN is annotated with (i) the schema instance that reduced (expanded) the node, (ii) its *e-conditions* (external effect conditions), the effects of any node below n in the hierarchy that support a validation outside of the n-subtree, (iii) its *e-preconditions*, the preconditions of any node in this subtree supported by a node outside of the subtree, and (iv) its *p-conditions* (persistence conditions). The latter, p-conditions, are validations whose source is scheduled before, and whose destination is scheduled after the task of node n, thus requiring that the validation's effect is not invalidated by any node within the subtree of n. This structure serves both to evaluate whether the plan is still valid and for replanning when it is not. A violation can be the failing of a validation, a missing validation, or an unnecessary validation. Roughly, in the two former cases a new sub-goal node for achieving the missing support is added to the network, in the latter case, the unnecessary validation is removed and this removal propagated, potentially removing any supporting actions which are no longer required. The resulting modified HTN will be handed back to the planner, reducing potentially remaining new, open sub-goals. Using the blocks world domain, Kambhampati shows that doing plan repair this way can be between 30% and 98% faster than planning from scratch depending on the similarity of the two problem instances. The results also show that plan modification generally seems to pay off more the more complex the problems are, in this case measured by the number of blocks. But the setup and generality of the results were questioned by [Nebel and Koehler, 1995] for two reasons:

1. all considered instances belong to a sub-class of planning problems in the blocks world domain which are solvable in polynomial time,

2. all instances are free of "deadlocks", meaning that it is never necessary to put a block temporarily on the table in order to reach the goal.

Based on the SNLP partial-order planner, [Hanks and Weld, 1995] implemented a plan modification system called SPA. They also annotate the steps of the plan with the "reasons" for adding it, but they utilize this information differently: when the need for plan modification arises, the information can be used to retract earlier additions with all their consequences. The presented replanning algorithm then just performs search in the space of partial plans starting from the current one. The presented empirical comparison to PRIAR draws a mixed picture: the savings rate of PRIAR is higher, but the authors argue that this is because PRIAR is simply slower in generative planning[5], a claim also supported by Nebel and Koehler. But also from a theoretical point of view PRIAR's superiority makes sense as PRIAR seems to exploit the plans annotation more.

[5]Both replanning systems are compared to their own generative planner to determine their relative savings.

Another partial-order planner based approach is the GPG system of [Gerevini and Serina, 2000] based on planning graphs [Blum and Furst, 1995]. The modification of a plan for altered initial and/or goal sets of atoms is driven by inconsistencies in the plan. An inconsistency is either an open precondition, an open goal condition, or two parallel actions that are mutually exclusive. The main algorithm (ADJUST-PLAN) processes these inconsistencies one by one starting with those of least time index. A window around the inconsistency is cut out of the plan and replanned, using the set of true fluents at the beginning of the window and the set of preconditions of actions at the end of the window as initial and goal states. If there is no plan for the current window, the window is increased. Replanning for one inconsistency may introduce new inconsistencies later in the plan. These are dealt with as the considered time index proceeds. The replanning algorithm is clearly sound and complete since in the worst case it ends up performing replanning from scratch, namely when the replanning window is spanning the entire plan. Surely, only replanning for the preconditions of the actions immediately to follow the window is not enough, as goal conditions may be affected by the replanning if e.g. an action inside the window that achieved a goal condition is not re-added and not compensated for otherwise. This insight led Gerevini and Serina to what they call the Backward Ω-goal set, which in fact is again nothing more than the regression of the goal over the remainder of the plan as we have seen it implicitly already in PLANEX. Ideally one would use this set as the sub-goal to plan for when cutting a window instead of just using the preconditions of actions immediately after. In the considered partial-order setup this set can however not be computed when there are still inconsistencies in the remainder of the plan in which case GPG approximates the set. Experimental results on slightly modified `logistics`, `rocket`, and `gripper` example problems show that this technique can again be much faster than planning from scratch, in this case up to four orders of magnitude.

An even more efficient and fairly universal approach to plan repair was recently presented at ICAPS. The idea in [van der Krogt and de Weerdt, 2005] is to reuse the heuristic used by a competitive planner to guide plan modification. The motivation for this is that replanning is not essentially different from generative planning, but still most available replanning systems do not use a competitive planner. Roughly, the idea is to nondeterministically remove actions from the plan and then add new actions, where the choice of actions to remove or add is guided by the heuristic. The presented approach is universally applicable with all heuristic planners but requires that the heuristic is capable of evaluating arbitrary partial plans. While this is not generally the case, as most heuristic planners (e.g. FF) deploy forward-search and their heuristics are designed accordingly, van der Krogt and de Weerdt propose the following method to overcome this problem: after removing an action from the plan, divide the remainder into pieces, so-called *cuts*, in such a way that no two actions in the same cut were previously connected by a removed action.[6] Then, from each cut a macro-action is created and added to the theory. After that, evaluating the empty plan, or,

[6]This is always the case in totally-ordered plans, but in the exemplified partial-order setting (using the VHPOP planner) plans can be graphs as actions can be performed concurrently and then this requirement makes sense.

in heuristic forward-search terminology, evaluating the initial state, provides the required heuristic information for the partial plan that was created from removing actions. An empirical comparison with GPG on problems of slightly modified initial states or goals shows that the presented approach can be anywhere between two to four times faster on some domains and twice as slow on others, while the plan quality, defined as the length, remains comparable. While the idea of lifting the planning heuristic to replanning is certainly an interesting new perspective, the automatic creation of macro actions from plan fragments is not always trivial depending on the action description language, putting a crimp in the universality argument.

One of the results in [Nebel and Koehler, 1995] was that plan reuse, and therefore also also plan repair, can be even less efficient than plan generation in the worst case when minimality of changes, so-called conservative plan modification, is required. Also this concern was addressed empirically: In their 2006 ICAPS paper [Fox et al., 2006] presented a thorough empirical comparison of plan repair and plan generation on PDDL 2.1 problems using the LPG planner, a local search heuristic planner similar to the earlier used GPG system. Fox et al. define the *distance* between two plans as the cardinality of actions appearing in either plan but not their intersection and speak of greater *plan stability* when one replanning strategy produces a new plan of a smaller distance to the old plan than another replanning strategy. To maintain high plan stability, they extend LPG's heuristic by a term penalizing the addition or removal of actions increasing this distance. Unsurprisingly the modified system achieves greater plan stability on replanning tasks than planning from scratch, while generally but not always being faster then replanning from scratch. However, Fox et al. use a different notion of conservative modification than the one underlying the results of Nebel and Koehler. Nebel and Koehler distinguished several ways of modifying a plan and were only able to show that conservative modification can be more complex than planning from scratch for modifications where not only the cardinality of the actions the two plans have in common is maximal, but also their order is the preserved. This restriction is not present in the work by Fox et al..[7]

In contrast [Sapena and Onaindia, 2002] adopt a strategy where actions can only be deleted at the front of a plan. The objective in this work is to find a plan suffix that is executable from a state which is reached after executing a minimal number of actions in the current (actual) state.[8] The presented but very limited experimental results show that this approach is faster than planning from scratch on problems with minor changes to the initial state while slower on problems with major changes.

In the robotics community search techniques based on the A^* algorithm dominate approaches used to address the navigation problem. This is the problem of navigating a robot through a dynamic environment without colliding with any objects. Due to the dynamics it is often the

[7]The modification strategy of Fox et al. corresponds to the MODMIX strategy of Nebel and Koehler for which they were not able to show above worst case results (cf. the footnote on page 9 of [Nebel and Koehler, 1995]).

[8]This is subsumed by the MODDEL strategy of Nebel and Koehler.

case that the presence or position of obstacles change[9] while a robot is executing a navigation plan and then the robot needs to replan its trajectory. This is a special case of the general replanning problem we are concerned with but here only the applicability of actions in the search tree change, or, in search terminology, edges and states are removed from or added to the search tree or the costs assigned to edges change. Despite these limitations the more advanced research results along these lines could serve as inspiration for our, more general, problem. These techniques all guarantee optimality[10] of the plan modification result and also other aspects have been addressed and now include, for instance, an anytime algorithm for this sort for "replanning" ([Ferguson et al., 2005]). [Koenig et al., 2002] describe how the techniques can be lifted to symbolic replanning, unfortunately without lifting the limitation to cases where only the applicability of actions (in the search tree!) have changed. While this is certainly too limited from our perspective the guaranteed optimality of the resulting plan still makes this interesting for us. It is important though to understand the complexity of the limitation. If none of the edges in the search tree is affected, the algorithm will do nothing and claim optimality of the current plan. However it is easy to construct examples where this fails, that is, cases where optimality is claimed but replanning from scratch would find a better plan. One is in settings with conditional effects: a discrepancy may not affect the preconditions or costs of any action in the plan, yet what the plan produces is not in accordance with what was planned for. Another way to confuse this approach is when the heuristic value depends on the available operators and their costs, as opposed to being a simple mapping from states (or state features) to numbers. In fact the heuristic functions most commonly used satisfy this criterion as they span a relaxed forward search graph to estimate a distance from the current state to the goal. During heuristic search planning, parts of the search space are pruned when, roughly, the heuristic function states that no plan of a better quality than x can be found in this part of the search space and there are candidates of quality better than x. But this information may change when the costs of operators in these pruned parts change, potentially making earlier pruned parts now more attractive. It is thus not enough to limit oneself to the edges in the originally spanned search tree, but the impact of discrepancies on the heuristic function has to be considered as well.

5.2 Backtracking

As an alternative to modifying the remainder of the current plan, people have also considered the possibility of on-line backtracking to previous points in the plan from where an existing alternative plan could be executed. This makes particular sense in conjunction with conditional planning. When a condition that decides between possible sub-plans is evaluated in a wrong belief about the actual state of the world, it may be beneficial to backtrack to this point of plan execution when later noticing the mistake, so that the correct sub-plan can be followed instead.

In [Golden et al., 1996] the motivation for backtracking is that under time constraints,

[9]more specifically the agent's belief about obstacles changes
[10]with respect to all possible plans in the current state

on-line systems may start executing a plan prefix while the plan has not been worked out completely yet. Backtracking to a previous choice point then becomes necessary when it turns out that the executed plan candidate does not reach the goal. Unfortunately, this paper, whose focus is the description of the XII planning system, only outlines the benefits and problems with backtracking. Recent work by [Eiter et al., 2004] is more elaborate. The authors propose the off-line generation of backtracking libraries that can be used as patch-plans upon execution failure to lead the system back to a diagnosed point of failure from where an alternative plan to reach the goal may be found. The key contribution of this paper is that of formulating the problem of finding pairs of action sequences and reverse plans as a conformant planning problem. From there, the authors show several complexity results by reduction from evaluation of Quantified Boolean Formulae (QBFs). In particular they show that determining whether a given action sequence has a reverse action or a reverse plan is Σ_2^p hard or Σ_3^p hard respectively (for the considered propositional action representation framework). In this work, a reverse action (resp. reverse plan) for an action sequence AS is, roughly, any action (resp. sequence of actions) such that for any two states S, S' for which executing AS in S produces S' it is the case that the reverse action (resp. plan) executed in S' always leads to S if it is executable in S'. This paper is only of theoretical interest: computing a reverse plan for every possible action sub-sequence of a plan prior to executing it does not seem practical considering the demonstrated complexity.

In [Soutchanski, 2003a] and his thesis ([Soutchanski, 2003b], Section 4.3.3, page 122), Soutchanski also proposes an extended recovery predicate involving backtracking. Unlike the recovery predicate of [De Giacomo et al., 1998], it is not required to find a repair (patch) plan with which the current remaining program can be prefixed to make it executable again, but it also considers backtracking to an earlier program state (recorded in a so-called program state history) from where alternative execution branches exist. The backtracking is realized through planning on-line. Again we note that Soutchanski's focus is on formalizing the problem in the Situation Calculus. In particular, the applied planning algorithms used to implement the specified predicates are poor compared to state of the art planning techniques. But this is not a limitation of Soutchanski's work, as his formal specifications of the task are independent of the methods applied in the implementations are in particular amenable to more sophisticated methods (within bounds).

5.3 Learning

In all approaches we have looked at so far one assumes that planning from scratch in the new, unexpected situation would produce a plan that is valid and optimal and in fact is taken as measure for any replanning algorithm in terms of quality and speed. But what if the discrepancy that occurred is due to a systematic error and will thus repeat itself? This is for instance the case when the agent applies an incorrect model of its own actions during planning. Consider the following example of a soccer robot equipped with a kicking device[11]: The user provided the robot with a model describing that kicking will make the ball travel

[11]This example is based on a true story.

in a straight line until it hits an obstacle. Unfortunately, during the game a fuse blows, causing the kicking device to fail entirely. Assume the robot has intercepted the ball and is in a good position to score a goal by either kicking or pushing the ball into the goal. Since kicking is usually faster this is the preferred option as it has a higher probability of success and so the robot triggers a kick action, but nothing happens, the kick is broken. The robot realizes that something went wrong when observing that the ball is still right in front of it as this observation is inconsistent with the model. But planning anew in the new situation will not do any good since the best plan will still be to kick instead of pushing the ball – according to the erroneous model. None of the approaches we have described so far would ever get out of this loop and the robot would miss its chance of scoring[12]. What is missing is a model adjustment step before replanning to account for modeling faults.

In this section we review replanning approaches where this problem has been addressed. We will elaborate on the obvious relation of this problem to reinforcement learning in the next section where we discuss other related fields of research.

[McNeill et al., 2003] consider the problems arising in the execution of agent plans in a multi-agent setting due to faulty ontologies. Their approach prescribes that before executing a plan, the plan is "deconstructed" in order to annotate it with the assumptions made during planning. These annotations are then used when a discrepancy arises, to pinpoint possible causes of the fault. Intuitively these annotations state why the agent thought the plan-elements would work, that is, why it thought each precondition was satisfied. If the action then fails at execution, one can conjecture what went wrong in the past, i.e. which past action failed to produce its required effect (compare this to the diagnosis task and in particular [Iwan, 2000]). At this point the paper makes a strong assumption namely that the reason why an action fails is given to the agent already as a condition, that is, if an action a fails the system is informed that this is because a – possibly yet unknown – precondition ϕ of a was not satisfied. The paper proposes to then correct the systems ontology and this may involve either changing facts in the theory, corresponding to changing the belief about the state of the world, or modifying the signature of the ontology itself. Unfortunately the paper does not give clues as to how one can decide what actually has to be changed and how this change can be done automatically.

Less original but more principled and detailed is the proposal of [Bjäreland, 1999]. This paper begins by formalizing the problem in the situation calculus and distinguishes between two sources for discrepancies *exogenous actions* (EA) and *violation of ontological assumptions* (VOA). The paper makes one critical assumption: the system is always able to tell whether an action has been executed completely or not by reading internal sensors. This is used to determine whether an EA or a VOA has caused a discrepancy: if no action has been executed but a discrepancy occurs it is assumed to be due to an EA, otherwise it is due to a VOA (combinations are not considered). When an EA occurs there is no need to adjust the model of the dynamics, instead just the current knowledge about the current situation is modified. This is done by replacing the axioms describing the initial situation S_0 with axioms describing the values of all ground fluents in the observed new state of the world. This is

[12]and finally impressing her developers

only possible in the face of complete knowledge about the initial situation. If the discrepancy is deemed to be due to a VOA, four cases are distinguished, two where the truth value of a fluent unexpectedly changed (positively or negatively), and two where it unexpectedly did not change (positively or negatively). In these cases the suggestion is to extend the successor state axioms, describing how fluents change in response to the execution of actions in the domain, according to the action that was performed and the discrepancy that was observed. This is done in a very obvious way: the conditions under which a fluent changes when the action in question is executed are either restricted by adding conjuncts or relaxed by adding disjuncts to the existing conditions. This does not seem to be a very favorable approach from the machine learning perspective as it does not perform any sort of generalization. While this may not be so critical with toy domains which can be modeled propositionally, in real-world systems the fluents of the model often involve many real-valued numbers, e.g. a position, and restricting learning to instances of these numbers will not be of any help in improving the model, as it is unlikely that the exact same state will be visited repeatedly. The assumption of perfect actuators, that is, assuming that the system always knows whether an actions has been executed completely, is problematic too. Some actions have indirect effects that only become observable sometime after the actual action was performed, but these effects would here be classified as exogenous and the system would thus never learn about their cause.

In [Wang, 1994] the author is concerned with learning planning operators by observing the actions of an expert agent and 'practicing' the newly acquired operators in the real-world to refine the model.[13] Observations consist of the state prior and the state after a named action is executed. The system is learning specific-to-general by simply generalizing preconditions and effects. Many assumptions are made: operators and states are deterministic, sensors are noise-free, the state is fully observable, preconditions are conjunctions of literals, and actions do not have conditional effects. In order to refine learned operators, the author proposes to solve practice problems in the environment to obtain more observations. While the paper does not elaborate on how to choose these practice problems, it does address the problem of planning using potentially over-constrained plan operators. When during planning the agent is uncertain about a conjectured but not yet verified precondition, the corresponding action may be considered applicable even when the conjectured precondition is not satisfied, in order to test the conjecture. A plan repair strategy that plans for open preconditions handles the potentially resulting execution failures.

Recently [Pasula et al., 2004] have addressed the same problem but in the presence of uncertainty about action outcomes. Given a set of (pre-state, action, post-state) examples, (s, a, s'), they greedily search for a best set of operators where the quality measure to maximize is the likelihood of the data given the operators, minus a term penalizing complex operator sets as determined through the number of preconditions and outcomes of all operators. The actual search is divided into three functions of which *LearnRules* is the main

[13]In fact, the author does not propose to practice in the real-world but a simulation thereof. This proposal though does not make much sense, because any simulation requires a model of the world itself. Thus if it is possible to implement and run such a simulator, the model learning problem has already been solved and the simulator model could be adopted by the agent. We thus think that practicing in the real-world is the only way to make sense out of this proposal.

function. It initializes the search by creating operators for each tuple (s, a) and then performs the search by specializing or generalizing these operators. Each time it creates a new operator it calls the second function *InduceOutcomes*. This function adds the specification of the outcomes (conjunctions of literals) to the operator. It again applies search to maximize the score (likelihood of data minus complexity) by merging compatible outcomes and removing redundant ones. Finally, the *LearnParameters* function adjusts the probabilities of these outcomes using a gradient method for optimization with respect to the score. While no theoretical properties like convergence guarantees or complexity are analyzed, the authors compare the approach to a learning method for Dynamic Bayesian Networks (DBNs) showing that the presented approach learns the true distribution of outcomes better than DBNs for all considered training set sizes. They also demonstrate that relational operators, i.e. operators with variable arguments, are learned faster than purely propositional (ground) ones.

Pasula et al.'s approach can be classified as learning from specific-to-general. Alternatively one can also learn from general-to-specific as demonstrated, for the purpose of learning planning operators under partial observability, in [Amir, 2004, Amir, 2005]. There the approach is to start out with the set of all possible transition systems and filter this set by a given sequence of action-observation tuples, only keeping those transition systems consistent with the observations. Dealing with the explicit set of all possible transition systems (called the *transition belief state*) is not feasible as it is doubly exponential in the number of domain features and the number of actions. Instead, Amir represents the transition belief more compactly as a formula of propositional logic. The actual learning of plan operators is then defined as the progression of this *transition belief formula* through the actions in the given sequence and the filtering by conjunction with the made observations. This approach will probably not be useful in our setting as we are starting with one specific transition system, the one we used in planning, and it is unclear how one could generate a larger set of possible transition systems from that in order to make this filtering approach applicable.

6 Related Work

In this section we briefly overview a few related areas of research. This presentation serves mainly to mark the boundaries of our survey and show the differences and similarities. In particular it does not claim to be a thorough summary of research in these related fields.

6.1 Decision Theory

In decision theory, people are concerned with optimally choosing actions in systems of stochastic state transitions in order to maximize a given utility function. The de-facto standard for representing these systems are Markov Decision Processes (MDPs) (see e.g. [Puterman, 1994, Boutilier et al., 1999]), when the state is fully observable, and Partially-Observable Markov Decision Processes (POMDPs), when the state is only partially observable. Solving an MDP (resp. POMDP) amounts to generating a *policy*, a control rule mapping states (resp. belief states) to actions in a way that maximizes the expected accumulated reward received from the states visited when following this rule (cf. Universal Plans in Section 3). Policies are universal. Since they map every state to an action, an agent executing the policy always knows what to do next and this choice will be optimal and so no execution monitoring for dealing with run-time discrepancies is required except for state estimation in the case of POMDPs. But the price for this is high as mentioned earlier in the context of Schopper's Universal Plans: the enumeration of all states makes the approach generally infeasible for large or infinite domains. Recent efforts try to work around this requirement by using compact representations [Hoey et al., 1999] and approximation techniques [St-Aubin et al., 2000]. Also First-Order MDPs are being investigated and a symbolic form of value iteration explored as a viable way for solving them [Boutilier et al., 2001]. First-Order MDPs are a promising approach, but the available solution techniques are not yet competitive with state of the art planners.

Further, while theoretically being the ultimate answer to run-time discrepancies in the current state of the system, policies become invalid when the goal, here the value function, changes. We will come back to this point in our final section.

When there remains uncertainty about the model applied in planning, there is a trade-off to be made. Either one exploits the current model, that is, tries to maximize the reward by behaving optimally according to it, or one decides to explore the environment in order to improve the model and benefit from this information gain in the future. This problem has been formalized and addressed in decision theory, reinforcement learning, and adaptive control (see for example [Duff, 2002], Chapter 2 for a survey). While the approaches have their appeal because of their rigorous mathematical foundation, they are limited in their expressiveness by using a transition function that maps ground states and ground actions to new states. For instance, a system performing action `pickup(a)` 1000 times does not learn anything about `pickup(b)`. This contrasts with the approaches presented in Section 5.3.

6.2 Metareasoning

In the face of limited computational resources[14], a rational agent interleaving planning and execution in a dynamic world should also be aware that deliberation itself impacts the state of the world. This is because the world evolves while the agent deliberates. In many applications it may thus be sometimes beneficial to commit to a seemingly sub-optimal plan quickly, because determining the optimal plan may just 'cost'[15] more than the potential gain, namely when time directly or indirectly affects the preferences of the agent. Reasoning about the agent's own reasoning process and capabilities is called *metareasoning* ([Russell and Wefald, 1991]). In the context of execution monitoring we may be faced with questions of metareasoning when monitoring plan optimality: It may be that the current plan, while still valid, has been found to be sub-optimal. Then it may still be optimal (in

[14]and this is the case for any real-world system

[15]We use the terms 'cost' and 'gain' loosely here and understand them to stand for any negative, resp. positive, impact on the agents preference criteria.

the meta sense) to continue its execution, namely when replanning will cost more than the potential gain. Another aspect of metareasoning concerns the evaluation step itself: if the evaluation incurs costs, e.g. by interrupting the execution, it may be beneficial to omit evaluation and any subsequent replanning entirely. [Russell and Wefald, 1991] suggest two main applications for metareasoning: (a) enabling the agent to decide on-line which computations to perform and which not, but also (b) analyzing the rationality of a system design. We can hope that in particular with respect to the latter, metareasoning can help lay the groundwork for evaluating execution monitoring designs and methods theoretically rather than just empirically.

6.3 Control Theory

The high-level objective of control theory – "control a system such that it behaves in a particular way" – is very similar to ours. The main difference between control theory and AI planning and execution monitoring approaches are the applied mathematics (cf. e.g. [Dean and Wellman, 1991]). Typically in control theory states are represented through the values of continuous variables, and 'control laws' map the current state and current time to control values, much like policies do in decision theoretic planning. While the dynamics of the controlled system can generally be characterized by algebraic, often differential, equations, the use of an explicit model is uncommon. Often the *error*, $e(t)$, the difference between the current state and the desired state at time t, is used directly to control the system. The very popular proportional-integral-derivative (PID) controller for example defines the control value to take in terms of a linear combination of the error itself, $e(t)$, the accumulated error, $\int_t e(t)$, and the error's gradient, $\frac{de(t)}{dt}$. Apart from the difference in applied techniques, research in control theory is also generally concerned with other questions such as the controllability, stability, or diagnosability of a system. Similarities exist in so-called optimal control which is concerned with optimizing the systems behavior with respect to some "cost index", the counterpart to preferences in AI planning. Finally, adaptive control techniques address the problem of refining the controller automatically, by adjusting the control parameters as seems necessary during operation.

7 Conclusion and Proposed Research

Revisiting the modules of the execution monitoring framework of Section 2 we make the following observations:

- Diagnosis: While approaches using numeric probabilities to reason about the most likely diagnosis are compelling from a mathematical point of view, they suffer from the problem of knowledge elicitation. In particular in complex dynamical systems, eliciting precise numeric probabilities from experts may be infeasible. As such, some form of qualitative probability measure is compelling. We argue that experts do not think about probabilities in numeric Markovian terms, but in temporally extended qualitative terms, like “After driving over peebles, there is an increased probability of erroneous odometry sensor readings.” or “When the CPU fan fails, it is likely that the CPU will overheat at some point.”

  The role of diagnosis in the context of execution monitoring also leaves some open questions. For instance, when a rationale-based approach to state evaluation is used and the plan has been annotated at each step with a condition for its validity, diagnosis should not provide the most likely diagnosis, but rather the probability for this condition being true.

- Evaluation: This part of the task has received surprisingly little attention in particular with respect to monitoring continued optimality of a plan. The only exception worthwhile noting, [Boutilier, 2000] (cf. page 19), makes strong assumptions about the available data, but may be a good starting point for further research. Nearly all approaches for evaluating discrepancies with respect to plan validity are based on the regressed goal, even though most authors do not seem to be aware of this. Yet no attempt has been made to lift this approach to evaluate discrepancies with respect to plan optimality.

- Replanning: Plenty of work exists regarding plan modification, most of it contributing an algorithm and an empirical analysis, showing its speed-up over planning from scratch, but with [Nebel and Koehler, 1995] (cf. page 20) theoretical investigations on this matter exist as well. However, virtually none of the existing approaches aims at maximizing plan optimality rather than a speedy reestablishment of plan validity. Exceptions are either limited in their applicability (cf. page 25, [Koenig et al., 2002]) or apply a specific preference criterion which is in particular distinct from the preferences that lead to the current plan ([Cushing and Kambhampati, 2005], [Fox et al., 2006], cf. page 24).

Also there are no satisfactory integrated approaches, i.e. execution monitoring systems applying state-of-the-art techniques in all of these parts. While modularizing has its obvious appeal, an integrated approach could have potential benefits as well. For instance it may not be necessary to disambiguate between two possible diagnoses when both faults provoked the same course of action, for instance the same repair plan, or both suggested replanning from scratch.

7.1 Proposed Research

Principally our interest lies with highly dynamic real-time domains of continuous state and action spaces, with lots of uncertainty, both exogenous and endogenous, and where specifying or learning a fail-proof model is not feasible or practical. We believe that in these cases, planning has to be paired with execution monitoring and an ideal execution monitor should:

- track the actual state of the world;

- determine whether the current plan has become sub-optimal or invalid and correct this;
- refine the model when necessary;
- operate on-line and not require significant changes to the applied planning method, in particular not require the off-line computation of a policy;
- allow for changing goals and preferences;
- be introspective and able to reason about the effects of its own deliberation processes (metareasoning);
- monitor replanning in order to react appropriately to discrepancies during that time as well.

In particular we believe that the following issues are worthwhile investigating:

1. Rationale-Based Monitoring: We believe that the rationale underlying the original construction of a plan is a good starting point in analyzing the impact run-time discrepancies have on the overall objective, namely reaching a given goal (plan validity) or reaching a goal in the best possible way (plan optimality). This belief is supported by several existing approaches but it has not been approached in a principled fashion yet and has still to be formalized. Also the use for monitoring optimality rather then just validity of a plan is a new and challenging aspect. Once formalized, the applicability of this approach for monitoring planning and execution of systems under various forms of soft and hard constraints and for various planning paradigms should be investigated. For instance, monitoring plan validity in the face of temporally extended goals would be an interesting aspect, but also monitoring plan optimality of conditional plans could be studied.

2. Rationale-Based Replanning: Similar to the last item, the formalization of the rationale may also be exploited in replanning, in particular for deciding whether or not plan-repair is likely to be more efficient than planning from scratch and, if so, the rationale my guide the repair. When Nebel and Koehler investigated the impact of problem similarity on the efficiency of plan-repair, they used a purely syntactic similarity measure. It remains to be shown that a more semantic, i.e. problem specific, similarity measure still not warrants better repair complexity. Such a similarity measure may be inspired by the rationale.

3. Conditional Planning versus Execution Monitoring: While not the main motivation for our work, it is also interesting to investigate how the trade-off between detailed modeling and subsequent conditional planning should be made against the alternative of naive modeling or little conditional planning (e.g. only for the most likely cases) and thus faster planning together with tight execution monitoring. Here, insights from metareasoning may be relevant.

4. Execution Monitoring for Changing Objectives: While most people consider execution monitoring a means for dealing with run-time discrepancies regarding the current state of the world, it can also happen that the agent's goal or preferences change. When such a change has not been anticipated and planned for, e.g. in form of a contingency plan, the system again has to decide how to react to this matter.

5. Preferred Dynamic Diagnosis: McIlraith ([McIlraith, 1997], cf. page 16) has cast the problem of dynamic diagnosis as a planning problem and recently we have investigated means for expressing complex qualitative temporal preferences in planning ([Bienvenu et al., 2006, Baier et al., 2007]). We believe these can be used to express complex qualitative temporal probabilities over possible diagnoses as well. These are easier to elicit from experts than numeric probabilities and overcome the Markovian restriction of standard probabilistic frameworks.

A long-term goal is to design a planning and execution system, very likely based on either the ReadyLog or IndiGolog framework, and demonstrate its applicability in some real-world domain with above characteristics.

References

[Ambros-Ingerson and Steel, 1988] Ambros-Ingerson, J. and Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI)*, pages 83–88.

[Amir, 2004] Amir, E. (2004). Learning partially observable action models. In *Proceedings of The 4th International Cognitive Robotics Workshop, at ECAI-2004, Valencia, Spain.*

[Amir, 2005] Amir, E. (2005). Learning partially observable deterministic action models. In *19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 1433–1439.

[Baier et al., 2007] Baier, J., Bacchus, F., and McIlraith, S. (2007). A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India. To appear.

[Baier and McIlraith, 2006] Baier, J. and McIlraith, S. (2006). Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI06)*, pages 788–795, Boston, MA.

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

[Beetz and McDermott, 1996] Beetz, M. and McDermott, D. (1996). Local planning of ongoing activities. In Drabble, B., editor, *Proceedings of the Third International Conference on AI Planning Systems*, pages 19–26, Morgan Kaufmann.

[Bienvenu et al., 2006] Bienvenu, M., Fritz, C., and McIlraith, S. A. (2006). Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, June 2006*, pages 134–144.

[Bjäreland, 1999] Bjäreland, M. (1999). Recovering from modeling faults in GOLOG. In *IJCAI'99 Workshop: Scheduling and Planning Meet Real-Time Monitoring in a Dynamic and Uncertain World, Stockholm, Sweden, August 1999*.

[Bjäreland, 2001] Bjäreland, M. (2001). *Model-Based Execution Monitoring*. PhD thesis, Linköping University, Schweden.

[Blum and Furst, 1995] Blum, A. and Furst, M. L. (1995). Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, Montréal, Québec, Canada. Morgan Kaufmann, 2 Volumes.

[Boutilier, 2000] Boutilier, C. (2000). Approximately optimal monitoring of plan preconditions. In *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI'00)*, pages 54–62, Stanford University, Stanford, California, USA. Morgan Kaufmann.

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

[Boutilier et al., 2001] Boutilier, C., Reiter, R., and Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence 2001*, pages 690–700.

[Colbry et al., 2002] Colbry, D., Peintner, B., and Pollack, M. E. (2002). Execution monitoring with quantitative temporal bayesian networks. In *Proc. of the 6th International Conference on AI Planning and Scheduling*, pages 194–203.

[Cushing and Kambhampati, 2005] Cushing, W. and Kambhampati, S. (2005). Replanning: A new perspective. In *Poster Program, ICAPS 2005*.

[de Freitas et al., 2004] de Freitas, N., Dearden, R., Hutter, F., Morales-Menendez, R., Mutch, J., and Poole, D. (2004). Diagnosis by a waiter and a mars explorer. Invited paper for Proceedings of the IEEE, Special Issue on Sequential State Estimation, 92(3):455-468.

[De Giacomo et al., 1998] De Giacomo, G., Reiter, R., and Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 453–465.

[de Kleer, 1991] de Kleer, J. (1991). Focusing on probable diagnoses. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'91)*, pages 842–848. AAAI Press / The MIT Press, ISBN 0-262-51059-6, Volume 2.

[de Kleer et al., 1992] de Kleer, J., Mackworth, A., and Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222.

[Dean and Wellman, 1991] Dean, T. and Wellman, M. (1991). *Planning and Control*. Los Altos, Calif.: M. Kaufmann Publishers.

[Dearden and Boutilier, 1994] Dearden, R. and Boutilier, C. (1994). Integrating planning and execution in stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, pages 55–61, Stanford, CA.

[Dix et al., 2003] Dix, J., Eiter, T., Fink, M., Polleres, A., and Zhang, Y. (2003). Monitoring agents using declarative planning. In *Proceedings 26th German Conference on Artificial Intelligence (KI 2003)*, pages 646–660, University of Hamburg, Germany. LNCS/LNAI 2821.

[Doyle et al., 1986] Doyle, R. J., Atkinson, D., and Doshi, R. (1986). Generating perception requests and expectations to verify the execution of plans. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI'86)*, pages 81–88, Philadelphia, PA, USA. Morgan Kaufmann, Two Volumes, Volume 1: Science.

[Duff, 2002] Duff, M. (2002). *Optimal Learning: Computational Procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts Amherst.

[Earl and Firby, 1997] Earl, C. and Firby, J. (1997). Combined execution and monitoring for control of autonomous agents. In *Proceedings of the First International Conference on Autonomous Agents*, pages 88–95, Marina del Rey CA, USA.

[Eiter et al., 2004] Eiter, T., Erdem, E., and Faber, W. (2004). Plan reversals for recovery in execution monitoring. In *Proceedings of The 10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, Whistler, Canada.

[Ferguson et al., 2005] Ferguson, D., Likhachev, M., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 262–271, Monterey, California, USA.

[Ferrein et al., 2004] Ferrein, A., Fritz, C., and Lakemeyer, G. (2004). On-line decision-theoretic Golog for unpredictable domains. In *Proceedings of The 4th International Cognitive Robotics Workshop, at ECAI-2004, Valencia, Spain.*

[Fichtner et al., 2003] Fichtner, M., Grossmann, A., and Thielscher, M. (2003). Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2–4):371–392.

[Fikes et al., 1972] Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288.

[Fox et al., 2006] Fox, M., Gerevini, A., Long, D., and Serina, I. (2006). Plan stability: Replanning versus plan repair. In *Proceedings of The International Conference on Automated Planning & Scheduling, Lake District, U.K., June 6–10.*

[Gerevini and Serina, 2000] Gerevini, A. and Serina, I. (2000). Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 112–121, Breckenridge, CO, USA.

[Golden et al., 1996] Golden, K., Etzioni, O., and Weld, D. (1996). Planning with execution and incomplete information. Technical Report 96-01, University of Washington.

[Hanks and Weld, 1995] Hanks, S. and Weld, D. S. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research (JAIR)*, 2:319–360.

[Hansen and Cohen, 1992] Hansen, E. A. and Cohen, P. R. (1992). Learning a decision rule for monitoring tasks with deadlines. Technical Report 92-90, Univ. of Massachusetts, Experimental Knowledge Systems Laboratory.

[Hoey et al., 1999] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 279–288, Stockholm, Sweden.

[Iwan, 2000] Iwan, G. (2000). Explaining what went wrong in dynamic domains. In *Proceedings of the 2nd International Cognitive Robotics Workshop.*

[Iwan and Lakemeyer, 2003] Iwan, G. and Lakemeyer, G. (2003). What observations really tell us. In *KI 2003: Advances in Artificial Intelligence*, pages 194 – 208.

[Kambhampati, 1990] Kambhampati, S. (1990). A theory of plan modification. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 176–182, Boston, Massachusetts. AAAI Press / The MIT Press, ISBN 0-262-51057-X, 2 Volumes.

[Koenig et al., 2002] Koenig, S., Furcy, D., and Bauer, C. (2002). Heuristic search-based replanning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS'02)*, pages 294–301, Toulouse, France.

[Lazovik et al., 2003] Lazovik, A., Aiello, M., and Papazoglou, M. (2003). Planning and monitoring the execution of web service requests. Technical Report DIT-03-049, Informatica e Telecomunicazioni, University of Trento.

[Lespérance et al., 2000] Lespérance, Y., Levesque, H., Lin, F., and Scherl, R. (2000). Ability and knowing how in the situation calculus. *Studia Logica*, 66(1):165–186.

[Levesque et al., 1997] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83.

[McCarthy, 1977] McCarthy, J. (1977). Epistemological problems of artificial intelligence. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI'77), invited talk*, pages 1038–1044, Cambridge, MA, USA. William Kaufmann.

[McIlraith, 2000] McIlraith, S. (2000). Diagnosing hybrid systems: A bayesian model selection approach. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX'00)*, pages 140–146.

[McIlraith et al., 2000] McIlraith, S., Biswas, G., Clancy, D., and Gupta, V. (2000). Hybrid systems diagnosis. In *Proceedings of Hybrid Systems: Computation and Control*, pages 282–295.

[McIlraith, 1997] McIlraith, S. A. (1997). Explanatory diagnosis: Conjecturing actions to explain obsevations. In *Proceedings of the Eighth International Workshop on Principles of Diagnosis (DX'97)*, pages 69–78.

[McNeill et al., 2003] McNeill, F., Bundy, A., and Schorlemmer, M. (2003). Dynamic ontology refinement. In *Proceedings of ICAPS'03 Workshop on Plan Execution*, Trento, Italy.

[Musliner et al., 1991] Musliner, D., Durfee, E., and Shin, K. (1991). Execution monitoring and recovery planning with time. In *Proceedings of the IEEE Seventh Conference on Artificial Intelligence Applications*, pages 385–388.

[Myers, 1998] Myers, K. L. (1998). Towards a framework for continuous planning and execution. In *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning.*

[Nebel and Koehler, 1995] Nebel, B. and Koehler, J. (1995). Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence (Special Issue on Planning and Scheduling)*, 76(1-2):427–454.

[Pasula et al., 2004] Pasula, H., Zettlemoyer, L. S., and Kaelbling, L. P. (2004). Learning probabilistic relational planning rules. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 73–82, Whistler, British Columbia, Canada.

[Pollack et al., 2003] Pollack, M. E., Brown, L. E., Colbry, D., McCarthy, C. E., Orosz, C., Peintner, B., Ramakrishnan, S., and Tsamardinos, I. (2003). Autominder: an intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, 44(3-4):273–282.

[Provan, 2005] Provan, G. (2005). Approximate model-based diagnosis using preference-based compilation. In *Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation (SARA)*, Airth Castle, Scotland, UK.

[Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes: Discrete Dynamic Programming*. Wiley, New York.

[Reiter, 1987] Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95.

[Reiter, 2001] Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge, MA.

[Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

[Russell and Wefald, 1991] Russell, S. and Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395.

[Sapena and Onaindia, 2002] Sapena, O. and Onaindia, E. (2002). Execution, monitoring and replanning in dynamic environments. In *AIPS-02 Workshop on On-line Planning and Scheduling*.

[Schoppers, 1987] Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. In McDermott, J., editor, *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 1039–1046, Milan, Italy. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Soutchanski, 2003a] Soutchanski, M. (2003a). High-level robot programming and program execution. In *Proceedings of the Workshop on Plan Execution, 10th June 2003, held at ICAPS'03, Trento, Italy*.

[Soutchanski, 2003b] Soutchanski, M. (2003b). *High-Level Robot Programming in Dynamic and Incompletely Known Environments*. PhD thesis, University of Toronto.

[St-Aubin et al., 2000] St-Aubin, R., Hoey, J., and Boutilier, C. (2000). APRICODD: Approximate policy construction using decision diagrams. In *Advances in Neural Information Processing Systems 13 (NIPS-2000)*, pages 1089–1095, Denver.

[van der Krogt and de Weerdt, 2005] van der Krogt, R. and de Weerdt, M. (2005). Plan repair as an extension of planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 161–170, Monterey, California, USA.

[Veloso et al., 1995] Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., and Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental AI*, 7(1).

[Veloso et al., 1998] Veloso, M. M., Pollack, M. E., and Cox, M. T. (1998). Rationale-based monitoring for continuous planning in dynamic environments. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98)*, pages 171–179, Pittsburgh, PA, USA.

[Verma et al., 2002] Verma, V., Fernandez, J., and Simmons, R. (2002). Probabilistic models for monitoring and fault diagnosis. In Chatila, R., editor, *The Second IARP and IEEE/RAS Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*.

[Wang, 1994] Wang, X. (1994). Learning planning operators by observation and practice. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS'94)*, pages 335–340, University of Chicago, Chicago, Illinois.

[Wilkins, 1985] Wilkins, D. E. (1985). Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33–45.

[Wilkins, 1988] Wilkins, D. E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers Inc., San Mateo, CA.

[Williams and Nayak, 1996] Williams, B. C. and Nayak, P. P. (1996). A model-based approach to reactive self-configuring system. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, volume 2, pages 971–978, Portland, Oregon, USA. AAAI Press / The MIT Press, 1996, ISBN 0-262-51091-X.

[Witteveen et al., 2005] Witteveen, C., Roos, N., van der Krogt, R., and de Weerdt, M. (2005). Diagnosis of single and multi-agent plans. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-05)*, pages 805–812, Utrecht, The Netherlands.