An Answer Set Programming Framework for Reactive Planning and Execution Monitoring (P-16536-N04)

D1: Report on State of the Art in Logic-Based Execution Monitoring

Esra Erdem, Thomas Eiter, Wolfgang Faber

Knowledge Based Systems Group,
Institute of Information Systems
Vienna University of Technology
Favoritenstraße 9-11, 1040 Vienna, Austria
(esra | eiter | faber)@kr.tuwien.ac.at

February 2004

1 Introduction

Planning has played an important role in many relevant areas of AI. The classical planning problem consists of the following task: Given a state of the world, several actions and their (deterministic) effects, find a sequence of actions (viz. a plan) to reach a certain goal state. In recent years, several successful logic-based approaches to classical planning have been proposed (cf. [33, 11, 63, 15]). In particular planning under the answer set programming (ASP) paradigm[1] has received considerable attention (cf. [46, 44, 59, 24, 22, 23, 18]).

Apart from classical planning, many suggestions have been made on how to plan under incomplete knowledge or with possible non-deterministic effects of actions, such as conformant planning [35], conditional planning [50] or universal planning [58]. All of these approaches concentrate on "statically" generating plans in advance. However, when planning for an autonomous agent or a group of collaborative agents in real world environments, more complex problems arise.

One cannot rely on a set of given initial states for which we can simply compute a plan and execute it without any interferences from outside the agents sphere of influence, which is the common assumption of most planners. In fact, exogenous events can disturb the plan and make it infeasible. Even worse, the assumed effects of an action can change due to some unforeseen reasons. For instance, assume a robot which is manually turned around while moving towards a target: It will, when sticking to its static plan, move away instead of getting closer to the target.

In this context the importance of the integration of reactive behavior with planning has been recognized as an emerging issue already more than a decade ago [32]: This is the point where "monitoring" and "belief revision" [31] come into play. Here, we still want to make use of plan generation in the classical sense, but add capabilities to revise or if necessary recompute these plans according to the current situation. In this context, revision can be seen twofold in this context by either trying to "repair" the plan at hand or by adapting the domain model we have used for generation of a failed plan and replan.

A further issue in collaborative environments can be the combination (plan merging/fusion, cf. [67]) of individual plans of collaborative agents. As described in Figure 1, agent planning in general is often described as a continuous loop of planning, acting and sensing the actual state of the world.

While logic programming methods and, in particular, ASP methods have proved to be profitably applicable for tasks such as knowledge-base updates (cf. [73, 39, 3, 2, 25, 26]), and diagnosis (cf. [53, 51, 13, 21]), to our knowledge the application to the special case of replanning and plan revision and

[1]For a comprehensive introduction to answer set programming we refer to [6] and to the proceedings of the "AAAI 2001 Spring Symposium on Answer Set Programming" [52] which was held in Stanford, CA in March 2001

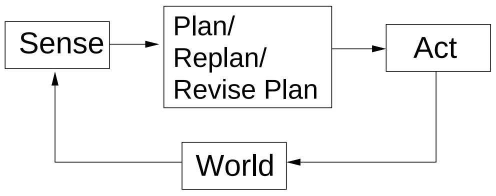

Figure 1: Sense-Act Loop with Planning

integration/interleaving with plan generation has not yet been examined extensively in a logic-based framework. This is one of the the main targets of our project.
Objective of this document. The objective of this document is to survey the state of the art in logic-based frameworks for execution monitoring by briefly comparing and positioning the work in this project to other related work, in order to provide a basis for drawing conclusions about the further course of the project, in order to clarify possibilities and restrictions.

The remainder of this document is structured as follows. In the next section, we consider execution monitoring of a single agent, while Section 3, we consider monitoring of multi-agent systems. Section 4 is devoted to the issue of domain revision for planning and action frameworks. After that, we turn in Section 5 to discuss possible directions of research. The final Section 6 reports on some activities which, in the light of the analysis, have been started and produced preliminary results already.

2 Monitoring with a Single-Agent

The earliest major treatment of execution monitoring was PLANEX [30]. It accepts goals from the user, calls the STRIPS planner to generate plans, and uses "triangle tables" for execution. PLANEX was used to control the robot Shakey.

The NASL planner [47] treats a planning problem as a specification for carrying out a complex action, so the planning and execution are completely unified.

IPEM [4] was the first system integrating partial-order planning with plan execution.

XFRM [9, 10] provides the continual modification of robot plans during their execution, using a rich collection of failure models and plan repair strategies. XFRM projects a default plan into its possible executions, diagnoses failures of these projected plans by classifying them into a taxonomy of predefined plan failures, and then revises the default plan by following the pointers from the predefined failures to predefined plan repair strategies.

Some other systems that integrate planning with execution monitoring are ROGUE [36], SIPE [70], SOAR [55], SPEEDY [8].

The idea of "universal plans" was developed in [58, 56, 57] to "address the tension between reasoned behaviour and timely response by caching reactions for classes of possible situations".

The recovery technique called "purely inserted recovery plans" is proposed in [48]. When an execution failure is detected, a new plan is inserted to be able to continue with the rest of the default plan.

In [37], monitoring tasks with deadlines is considered as a sequential decision problem, which makes available a dynamic programming method for constructing a decision rule for monitoring. However, the authors do not consider any recovery technique besides the decision to abandon a monitored task if it is about to miss a deadline.

The "rationale-based monitoring", based on the idea of planning as decision making, is introduced in [69]. The idea is to monitor sensing relevant (or potentially relevant) features of the world that are likely to affect the plan execution. Moreover, it investigates the balance between sensitivity to changes in the world and stability of the plans.

In this project, we are interested in logic-based monitoring frameworks. The existing logic-based frameworks for execution monitoring are presented in [12, 61, 62], and [28, 29].

In [12, 61, 62], execution monitoring is defined as "the robot's process of observing the world for discrepancies between the actual world and its internal representation of it, and recovering from such perceived discrepancies". The authors assume that the discrepancies are caused by exogenous actions and, moreover, the robot observes all such actions. In these frameworks, the agent detects a discrepancy if the current situation ("physical reality") is different from the expected situation ("mental reality"). If the discrepancy is "relevant", i.e., if the remaining plan (or, in general, program) is not successful, the agent tries to recover from it. For recovery, in [12], a new plan is computed so that executing it followed by the remaining plan would lead to a goal situation from the current situation. In [61], the authors consider "restartable plans" so that the agent can backtrack to a past nondeterministic choice point without having to compute a plan. After identifying the latest nondeterministic choice point, the agent executes the plan from that point on, to reach a goal situation from the current situation. If the agent cannot reach a goal situation then she identifies the next past nondeterministic choice point, and follows the recovery procedure as above. In [62], backtracking is considered in connection with inserting corrective plans as in [12], by a recursive recovery procedure like the one in [61]. The agent computes a plan from the current situation to reach the latest nondeterministic choice point. If executing the plan from that point on does not lead to a goal situation then the agent tries to recover by inserting a corrective plan at that point. If the agent cannot find a corrective plan then she finds the next past nondeterministic choice point, and follows the recovery procedure as above. In the framework of [62], the author considers sensing actions to find the real effects of actions.

In [28, 29], the notion of execution monitoring above is extended by that "the robot should come up with explanations for detected discrepancies". Here, the monitoring agent detects a discrepancy when the action is not executable or when the effects of an action are not as expected. Therefore, like in action monitoring, the detected discrepancies may not be relevant to the successful execution of the rest of the plan. Like in [62], the authors consider sensing actions to find the real effects of actions. After a discrepancy is detected, the monitoring agent provides an explanation for it. This is achieved by introducing abnormality predicates to describe the unintended or unexpected effects of actions. If an explanation is found then some predefined plans are executed to achieve the goals; otherwise, a new plan is computed to reach a goal situation from the current situation.

As for representation, in [12, 61, 62], execution monitoring is described in the situation calculus as in [54], which makes these approaches applicable to Golog programs [45]. In [29], execution monitoring is described in the fluent calculus [65] for FLUX programs [66]. These representations allow the authors of [12, 61, 62, 29] to show the usefulness of their approaches on real robots.

3 Monitoring with Multi-Agents

Teamwork monitoring has been recognized as a crucial problem in multi-agent coordination. Jennings [41] has proposed two foundations of multi-agent coordination: "commitments" and "conventions". Agents make commitments. Conventions are a means of monitoring of the commitments. The monitoring rules, i.e. what kind of information monitored and the way to perform monitoring, are decided by conventions. Jennings illustrates the method by some examples. However, he doesn't investigate how to select such conventions.

Myers [49] has introduced a continuous planning and execution framework (CPEF). The system's central component is a plan manager, directing the processes of plan-generation, -execution, -monitoring, and -repair. Monitoring of the environment is carried out at all time during plan generation and execution. Furthermore, execution is tracked by the plan manager by comparing reports of individual actions' outcomes with the temporal ordering relationships of actions. Several types of event-response rule have been concerned: (1) *failure monitors* encode suitable responses to potential failures during plan execution, (2) *knowledge monitors* detect the availability of information required for decision making, and (3)*assumption monitors* check whether assumptions that a given plan relies on still hold. The idea of assumption monitors helps early detection of potential problems before any failure occurs.

Based upon CPEF, Wilkins et al. have presented a system in [71]. The execution monitoring of agent teams is performed based on communicating state information among team members that could be any combination of humans and/or machines. Humans make the final decision, therefore, even if unreliable communications exist, the monitoring performance may not be degraded much with the help of humans experience.

Another interesting monitoring approach in multi-agent coordination is based on plan-recognition, by Huber [38], Tambe [64], Intille and Bobick [40], Devaney and Ram [14], Kaminka et al. [42, 43]. In this approach, an agent's intentions (goals and plans), beliefs or future actions are inferred through observations of another agent's ongoing behavior.

Devaney and Ram [14] have described the plan recognition problem in a complex multi-agent domain involving hundreds of agents acting over large space and time scales. They use pattern matching to recognize team tactics in military operations. The team-plan library stores several strategic patterns which the system needs to recognize during the military operation. In order to make computation efficient, they utilize representations of agent-pair relationships for team behaviors recognition.

Intille and Bobick [40] have constructed a probabilistic framework that can represent and recognize complex actions based on visual evidence. Complex multi-agent action is inferred using a multi-agent belief network. The network integrates the likelihood values generated by several visual goal networks at each time and returns a likelihood that a given action has been observed. The network explicitly represents the logical and temporal relationships between agents, and its structure is similar to a naive Bayesian classifier network structure, reflecting the temporal structure of a particular complex action. Their approach relies on all coordination constraints among the agents. Once an agent fails, it may not be able to recognize the plans.

Another line of work has been pursued in ISI. Gal Kaminka et al. [42, 43] have developed the *OVERSEER* monitoring system, which builds upon work on multi-agent plan-recognition by [40] and [64]. They address the problem of many geographically distributed team members collaborating in a dynamic environment. The system employs plan recognition to infer the current state of agents based on the observed messages exchanged between them. The basic component is a probabilistic plan-recognition algorithm which underlies the monitoring of a single agent and runs separately for each agent. This algorithm is built under a Markovian assumption and allows linear-time inference. To monitor multiple agents, they utilize social knowledge, i.e. relationships and interactions among agents, to better predict the behavior of team members and detect coordination failures. *OVERSEER* supports reasoning about uncertainty and time, and allows to answer queries related to the likelihood of current and future team plans.

4 Domain Revision

In the case of flaws or inconsistencies, one may need to modify the domain description as well. A related work of domain revision in the context of logic-based execution monitoring is presented in [28, 29].

In [28, 29], the authors try to distinguish between the recent and the outdated information using "knowledge update axioms" (KUAs). Incomplete knowledge of the world is represented by possible states. When a robot gains more information using its sensors, the set of possible states can be reduced to a smaller set satisfying this new information. KUAs update the set of possible states as follows. The fluents that are treated as dynamic are tagged with the time of their observations. For instance, for a fluent $f(\vec{x})$ and a time t, if $f(\vec{x}, t)$ is true in all possible states (of a situation) then this means that $f(\vec{x})$ has been observed to be true at time t. Such a time point is unique for each fluent at each situation. When a robot senses information about a dynamic fluent f at time t, it discards the previous knowledge on f and updates the set of possible states by recording f to be true (false, resp.) at all possible states depending on whether f holds (does not hold, resp.) in situation s. This allows the robot to forget some information depending on the value of t. Also, with many observations over time, the robot can gain sufficient confidence about a certain property and modify her belief about that property.

5 Directions for Research

5.1 Monitoring with a single-agent

We want to find a general logic-based framework for monitoring. We consider monitoring as a three step process, like in [28, 29]:

- detection of discrepancies,
- finding diagnoses of these discrepancies, and
- recovering from the detected discrepancies using these diagnoses.

Like [12, 61, 62], we want to give a formal specification of each step.

In [12, 61, 62], all possible trajectories of the monitored plan are taken into account for discrepancy detection; in [28, 29] no trajectories are considered. We consider monitoring relative to a given set of intended trajectories, which is more general than the above.

In [28, 29], the authors introduce abnormality predicates to find explanations for detected discrepancies. This approach is similar to the kind of diagnosis discussed in [7] and [5] where the authors detect faulty components of a system using abnormality predicates. We want to find diagnoses for discrepancies without introducing such predicates where we do not have to formulate all possible abnormal cases of the execution for each action.

Plan recovery by backtracking is not studied in [12, 28, 29], and it is studied to some extent in [61] (i.e., for restartable programs) and in [62] (i.e., backtracking as planning) using "traces" of the plan. We want to find a backtracking method for plan recovery that requires less amount of planning, and that does not require an execution history. In particular, our idea is to consider “reversing” actions for backtracking, as an (in general) incomplete but efficient method.

The idea of backtracking for recovery that we consider is similar to “reverse execution” in program debugging [72, 1], where every action is undone to reach a “stable” state. However, reverse execution in program debugging requires an execution history.

According to the framework we study, in the process of monitoring the execution of a plan relative to a set of intended trajectories, the monitoring agent

1. checks whether there is a discrepancy between the current state and the corresponding states of the given trajectories relative to the plan;

2. if no discrepancy is detected then continues with the execution of the plan; otherwise, tries to find a diagnosis of discrepancies by examining the given trajectories against possible evolutions of the current state from the initial state;

3. if a diagnosis is found then recovers from the discrepancies by backtracking to the diagnosed point of failure and executing the plan from that point on; otherwise, finds another plan from the current state to reach a goal state.

As for representation, in order to abstract from the details of a logic-based action representation framework like $\mathcal{K}$ or $\mathcal{C}$, we plan to consider the action representation framework of [68] to describe execution monitoring. In this representation framework, an action description can be represented by a transition diagram—a directed graph whose nodes correspond to states and whose edges correspond to action occurrences. It can accommodate nondeterminism, concurrent actions and dynamic worlds. Such action representations can be obtained from domain descriptions in STRIPS-based languages or in more expressive action description languages, such as $\mathcal{C}+$ [34] or $\mathcal{K}$ [22]. This allows one to employ systems like CCALC and DLV$^{\mathcal{K}}$, eventually.

We have started describing the monitoring framework above in [19] and [20]. One possible research direction is extending the monitoring framework with sensing actions for conditional planning, and with probabilistic information for quantitative uncertainty. This requires the extension of the underlying action description languages like $\mathcal{K}$ (see also section 5.3).

Another research direction is the update of the domain description in the case of flaws or inconsistencies. This may require the revision of the plan as well. For that, we need to consider a more general process than the third step of our framework above, which involves both patch planning and backtracking in general.

5.2 Monitoring with multi-agents

One interesting problem is, given a (possibly incomplete) description of a multi-agent system (MAS), to check whether the actual agent collaboration is compliant with the description. We have studied this problem in [16, 17]. Here, agent collaboration can be described as an action theory. Action sequences reaching the collaboration goal can be computed by a planner, whose compliance with the actual MAS behavior allows to detect possible collaboration failures. The approach can be fruitfully applied to aid

(1) debugging offline an implemented MAS, and

(2) monitoring online the collaboration of multiple agents.

The plan-recognition approaches described above mainly aim at inferring (sub-)team plans and future actions of agents. They do not address the MAS debugging issue. Furthermore, our approach above might be used in the MAS design phase to support protocol generation, i.e., determine at design time the messages needed and their order, for a (simple) agent collaboration. The agent developer may select one of the possible plans, e.g. according to optimality criteria such as least cost, P^*, and program the individual agents to obey the corresponding protocol. In subsequent monitoring and testing, P^* is then the (single) intended plan.

Plan recognition is suitable for various situations: if communication is not possible, agents exchanging messages are not reliable, or communications must be secure. It significantly differs from our approach in the following points:

1. If a multi-agent system has already been deployed, or it consists of legacy code, the plan-recognition approach can do monitoring without modifications on the deployed system. Our method entirely relies on an agent message log file.

2. The algorithms developed in [43] and [14] have low computational complexity. Especially the former is a linear-time plan recognition algorithm.

3. Our model is not yet capable of reasoning about uncertainty, time and space.

4. In some tasks, agents do not frequently communicate with others during task execution. In addition, communication is not always reliable and messages may be incorrect or get lost.

We believe the first three points can be taken into account in our framework:

(1) Adding an agent actions log file explicitly for a given MAS should not be too difficult.

(2) While the developed algorithms are of linear complexity, the whole framework needs to deal with uncertainty or probabilistic reasoning which can be very expensive. While our approach is NP-hard in the worst case, we did not encounter any difficulties in the scenarios we have dealt with.

(3) Although IMPACT (the agent system used in our framework) does not yet have implemented capabilities for dealing with probabilistic, temporal and spatial reasoning, such extensions have been developed and are currently being implemented.

5.3 Language Extensions

As for language extensions, we see that there are in principle two kinds:

- The first kind of extension is concerned with enriching the basic language, by constructs which on the one hand enlarge the scope of problems which can be expressed in dependent of monitoring and, on the other hand, might be profitably used for the monitoring process.
- The second kind of extension is tailored for the monitoring process, and contains constructs which only serve this purpose.

As for the first kind of extensions, the current language of the DLV$\mathcal{K}$ planning system, language $\mathcal{K}$, does not have sensing primitives to read values from the environment. The capability of processing sensory input would be very valuable, and thus an extension of the language $\mathcal{K}$ in this direction seems meaningful. In particular, if the behavior of a monitoring agent should be formulated itself as a (simple) action theory, then sensing actions would be mandatory for such a formalization.

Along with sensing, more flexible notions of plans than optimistic or conformant plans such as conditional plans would be an interesting extension of the basic language itself. However, the search space for conditional plans is huge, and only conditional plans of restricted form might be generated automatically. To our knowledge, in the area of logic-programming based planning this problem has been addressed only very recently [60].

Another direction is an extension of the language by taking besides qualitative uncertainty also quantitative uncertainty into account. In particular, probabilistic extensions of the language $\mathcal{K}$ would be interesting to have. However, the integration of qualitative and quantitative uncertainty in a single framework is not an easy task, and its computation might be complex. We can take profit of the preliminary work [27] on a probabilistic extension of the language $\mathcal{C}+$, which however needs to be adapted and refined.

For the second kind of extensions, language constructs might be developed which support a declarative specification of monitoring policies. An important notion in this context are checkpoints, i.e., points in time at which the real execution of a plan should be checked against the intended one. To this end, checkpoints have to be singled out in a way. A possible extension of the planning formalism could provide constructs for declaring checkpoints. This could be, e.g., on the basis of temporal information such as predefined points in time (e.g., also periodic time points), or in terms of conditions which need to be met.

Suitable constructs for expressing such checkpoint information could provide an element for a domain-specific declarative action language for monitoring, which builds on $\mathcal{K}$. To our knowledge, no similar language exists for logic-programming based planning systems.

6 Conclusion

The main target of this project is to integrate monitoring with planning in a general logic-based framework, and to extend the framework possibly using relevant methods from the areas of diagnosis and belief revision.

We have introduced a novel monitoring framework in [19, 20] for single agents. Different from the other logic-based frameworks described in Section 2, this framework should provide explanations, during the execution of a plan, for what goes wrong and when; it should not be necessary to identify a priori what may go wrong by introducing abnormality predicates for each action. To identify a possible point of failure, the monitoring agent should not have to consider all feasible trajectories for the plan, but only a given set of trajectories describing some "intended" or "preferred" executions of the plan. Then our framework can suggest a recovery by backtracking to a point of failure, which sometimes prevents replanning.

One interesting problem is, given a (possibly incomplete) description of a multi-agent system (MAS), to check whether the actual agent collaboration is compliant with the description. We have studied this problem in [16, 17]. The plan-recognition approaches described in Section 3 mainly aim at inferring (sub-)team plans and future actions of agents. They do not address the MAS debugging issue.

Extension of the monitoring framework for a single agent and extension of the action description language $\mathcal{K}$ accordingly remains as a future work, as discussed Section 5.1.

References

[1] H. Agrawal, R. A. DeMillo, and E. H. Spafford. An execution backtracking approach to program debugging. *IEEE Software*, 8(3):21–26, 1991.

[2] J. Alferes and L. Pereira. Logic Programming Updating - A Guided Approach. In A. Kakas and F. Sadri, editors, *Computational Logic: From Logic Programming into the Future*, volume 2408 of *Lecture Notes in AI (LNAI)*, pages 382–412. Springer Verlag, 2002. Festschrift in honour of Bob Kowalski.

[3] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic Updates of Non-Monotonic Knowledge Bases. *Journal of Logic Programming*, 45(1–3):43–70, 2000.

[4] J. Ambrose-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proc. AAAI-88*, pages 735–740, 1988.

[5] M. Balduccini and M. Gelfond. Diagnostic reasoning with a-prolog. *Journal of Theory and Practice of Logic Programming*, 3(4–5):425–461, 2003.

[6] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press, 2003.

[7] C. Baral, S. A. McIlraith, and T. C. Son. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Principles of Knowledge Representation and Reasoning*, pages 311–322, 2000.

[8] C. Bastié and O. Régnier. Speedy: Monitoring the execution in dynamic environments. In *Proc. of the Workshop at International Conference on Formal and Applied Practical Reasoning*, 1996.

[9] M. Beetz and D. McDermott. Improving robot plans during their execution. In *Proc. AIPS-94*, pages 3–12, 1994.

[10] M. Beetz and D. McDermott. Expressing transformations of structured reactive plans. In *Proc. ECP-97*, pages 66–78, 1997.

[11] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via Model Checking: A Decision Procedure for AR. In *Proceedings of the 4th Eutopean Conference on Planning (ECP-97)*, pages 130–142. Springer Verlag, 1997.

[12] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning*, pages 453–465, 1998.

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

[14] M. Devaney and A. Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proc. AAAI/IAAI-98*, 1998.

[15] Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding Planning Problems in Nonmonotonic Logic Programs. In *Proceedings of the European Conference on Planning 1997 (ECP-97)*, pages 169–181. Springer Verlag, 1997.

[16] J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang. Monitoring agents using declarative planning. In *Proc. KI-03*, 2003.

[17] J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang. Monitoring agents using declarative planning. *Fundamenta Informaticae*, 57(2–4):345–370, 2003.

[18] J. Dix, U. Kuter, and D. Nau. Planning in answer set programming using ordered task decomposition. *Journal of the Theory and Practice of Logic Programming*, October 2002. Revised paper, under review for a special issue on the ASP 01 meeting.

[19] T. Eiter, E. Erdem, and W. Faber. Finding Explanations for Discrepancies in Plan Execution. Submitted for publication, Jan. 2004.

[20] T. Eiter, E. Erdem, and W. Faber. Plan reversals for recovery in execution monitoring. Manuscript, 2004.

[21] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The Diagnosis Frontend of the `dlv` System. *AI Communications – The European Journal on Artificial Intelligence*, 12(1–2):99–111, 1999.

[22] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning, II: The DLV$^{\mathcal{K}}$ system. *Artificial Intelligence*, 144(1–2):157–211, 2002.

[23] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Answer Set Planning under Action Costs. *Journal of Artificial Intelligence Research*, 19:25–71, 2003.

[24] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. *ACM Transactions on Computational Logic*, 5(2), Apr. 2004.

[25] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. A Framework for Declarative Update Specifications in Logic Programs. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 649–654. Morgan Kaufmann, 2001. Extended version Technical Report INFSYS RR-1843-02-07, TU Wien, 2002.

[26] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On Properties of Update Sequences Based on Causal Rejection. *Journal of the Theory and Practice of Logic Programming*, 2(6):721–777, 2002.

[27] T. Eiter and T. Lukasiewicz. Probabilistic reasoning about actions in nonmonotonic causal theories. In C. Meek and U. Kjærulff, editors, *Proceedings Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003), August 7-10, 2003, Acapulco, Mexico*, pages 192–199, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[28] M. Fichtner, A. Großmann, and M. Thielscher. Intelligent execution monitoring in dynamic environments. In *Proc. of IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating*, 2003.

[29] M. Fichtner, A. Großmann, and M. Thielscher. Intelligent execution monitoring in dynamic environments. *Fundamenta Informaticae*, 57(2–4), 2003.

[30] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[31] D. Gabbay and Ph.Smets, editors. *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume III: Belief Change. Kluwer Academic, 1998.

[32] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence*, pages 677–682, Seattle, WA, 1987.

[33] E. Giunchiglia. Planning as satisfiability with expressive action languages: concurrency, constraints and nondeterminism. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pages 657–666, 2000.

[34] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

[35] R. Goldman and M. Boddy. Expressive planning and explicit knowledge. In *Proceedings AIPS-96*, pages 110–117. AAAI Press, 1996.

[36] K. Haigh and M. Veloso. Interleaving planning and robot execution for asynchronous user requests. In *Proc. IROS-96*, pages 148–155, 1996.

[37] E. Hansen and P. Cohen. Learning a decision rule for monitoring tasks with deadlines. Technical Report EKSL 92-80, Dept. of Comp. Sci., U. of Massachusetts at Amherst, 1992.

[38] M. J. Huber and E. H. Durfee. On acting together: Without communication. In *Spring Symposium Working Notes on Representing Mental States and Mechanisms*, pages 60–71, 1995.

[39] K. Inoue and C. Sakama. Updating Extended Logic Programs through Abduction. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 1999*), volume 1730 of *Lecture Notes in AI (LNAI)*, pages 147–161. Springer, 1999.

[40] S. S. Intille and A. F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proc. AAAI/IAAI-99*, 1999.

[41] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[42] G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring deployed agent teams. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, pages 308–315. ACM, 2001.

[43] G. A. Kaminka and M. Tambe. Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[44] N. Leone, R. Rosati, and F. Scarcello. Enhancing Answer Set Planning. In A. Cimatti, H. Geffner, E. Giunchiglia, and J. Rintanen, editors, *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, pages 33–42, Aug. 2001.

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

[46] V. Lifschitz. Answer Set Planning. In D. D. Schreye, editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP'99)*, pages 23–37, Las Cruces, New Mexico, USA, Nov. 1999. The MIT Press.

[47] D. McDermott. Planning and acting. *Cognitive Science*, 2(2):71–109, 1978.

[48] D. Musliner, E. Durfee, and K. Shin. Execution monitoring and recovery planning with time. In *Proc. IAIA-91*, 1991.

[49] K. Myers. CPEF: A continuous planning and execution framework. *AI Magazine*, 20(4):63–69, 1999.

[50] M. A. Peot and D. E. Smith. Conditional Nonlinear Planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 189–197. AAAI Press, 1992.

[51] D. Poole. Normality and Faults in Logic-Based Diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1304–1310, Detroit, Michigan, USA, 1989.

[52] A. Provetti and S. T. Cao, editors. *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning.* AAAI Press, Stanford, CA, March 2001.

[53] R. Reiter. A Theory of Diagnosis From First Principles. *Artificial Intelligence*, 32:57–95, 1987.

[54] R. Reiter. *Knowledge in action: Logical Foundations for specifying and implementing dynamical systems.* MIT Press, 2001.

[55] P. Rosenbloom, J. Laird, and A. Newell. *The SOAR Papers: Readings on Integrated Intelligence.* MIT Press, 1993.

[56] M. Schoppers. In defense of reaction plans as caches. *AI Magazine*, 10(4):51–60, 1989.

[57] M. Schoppers. Building monitors to exploit open-loop and closed-loop dynamics. In *Proc. AIPS-92*, pages 204–213, 1992.

[58] M. J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI) 1987*, pages 1039–1046, Milan, Italy, Aug. 1987. Morgan Kaufmann Publishers, Inc.

[59] T. C. Son, C. Baral, and S. McIlraith. Extending Answer Set Planning with Sequence, Conditional, Loop, Non-Deterministic Choice, and Procedure Constructs. In A. Provetti and S. T. Cao, editors, *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 202–209, Stanford, CA, March 2001. AAAI Press.

[60] T. C. Son, P. H. Tu, and C. Baral. Planning with sensing actions and incomplete information using logic programming. In V. Lifschitz and I. Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, pages 261–274. Springer, 2004.

[61] M. Soutchanski. Execution monitoring of high-level temporal programs. In *Proc. of IJCAI Workshop on Robot Action Planning*, 1999.

[62] M. Soutchanski. High-level robot programming and program execution. In *Proc. of ICAPS Workshop on Plan Execution*, 2003.

[63] V. Subrahmanian and C. Zaniolo. Relating Stable Models and AI Planning Domains. In L. Sterling, editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 233–247, Tokyo, Japan, June 1995. MIT Press.

[64] M. Tambe. Tracking dynamic team activity. In *Proceedings 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 80–87, 1996.

[65] M. Thielscher. The concurrent, continuous Fluent Calculus. *Studia Logica*, 67(3):315–331, 2001.

[66] M. Thielscher. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 2004. To appear.

[67] H. Tonino, A. Bos, M. de Weerdt, and C. Witteveen. Plan coordination by revision in collective agent based systems. *Artificial Intelligence*, 142(2):121–145, 2002. International Conference on MultiAgent Systems 2000.

[68] H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *Proc. of Eighth European Conf. on Logics in Artificial Intelligence (JELIA'02)*, pages 111–124, 2002.

[69] M. M. Veloso, M. E. Pollack, and M. T. Cox. Rationale-based monitoring for planning in dynamic environments. In *Proceedings 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 171–180, 1998.

[70] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, 1988.

[71] D. Wilkins, T. Lee, and P. Berry. Interactive execution monitoring of agent teams. *Journal of Artificial Intelligence Research*, 18:217–261, 2003.

[72] M. Zelkowitz. Reversible execution. *Communications of ACM*, 16(9):566, 1973.

[73] Y. Zhang and N. Y. Foo. Updating Logic Programs. In H. Prade, editor, *Proceedings 13th European Conference on Artificial Intelligence* (*ECAI 1998*), pages 403–407. Wiley, 1998.