A Self-Help Guide For Autonomous Systems

Michael L. Anderson, Scott Fults, Darsana P. Josyula, Tim Oates,
Don Perlis, Matthew D. Schmill, Shomir Wilson, and Dean Wright

1 Introduction

Things almost never go exactly as we plan or hope; deviations from the ideal are a fact of life. As Heraclitus noted almost 2,500 years ago, "only change endures". Though Heraclitus was silent on whether the inevitable change will be good or bad, Murphy's Law tells us to expect the worst. Given the pervasive and often perverse nature of the unexpected, it is not surprising that humans are adept at dealing with it—we would not have survived and flourished otherwise. This facility with the unexpected may account for our annoyance when our automated systems break in situations that we could deal with easily. It is then we need to remember that such failures account for much of the progress in AI, since—as this special issue attests—these annoying surprises goad us into insights and thus to improvements in the underlying algorithms.

But one can also view this phenomenon in another light. Instead of our having to rebuild the algorithms when—in an unanticipated situation—they make "stupid" mistakes that we could easily have dealt with, why not build them to do what we do: deal with the mistakes on their own? This is a tall order, and not a new one: it requires addressing the so-called brittleness problem (cf. [5]). In this article, we present our own approach to this problem. First we describe the conceptual focus—which we call the metacognitive loop (MCL)—that is guiding our work; then we present a number of domain-specific implementations of MCL, which we think provide significant evidence that MCL is a general-purpose methodology for building flexible (non-brittle) programs in specific domains; and finally we discuss our current efforts toward a domain-independent implementation of MCL.

2 MCL Systems

Our starting point is to ask what mechanisms humans use when dealing with the unexpected. Our contention is that people learn what to expect, notice when an expectation fails, and then decide what to do about it.[1] More specifically humans tend to (and machines ought to):

N: Note when an anomaly occurs

A: Assess it

G: Guide a response into place.

Looping through this NAG cycle—the basic form of MCL—leads to remarkably positive results when people do it, from noting a successful behavior in one domain (e.g., swinging a cricket bat) utterly fails in another (baseball) and is best jettisoned and a new one learned from scratch rather than tweaking the old, to noting and correcting a misunderstanding in conversation, to trying a new approach to a problem (e.g., on the final exam in light of failures on the midterm). Moreover, asking for help or giving up are in fact among the most important things in the human repertoire of repairs, for, as the saying goes: if you find yourself in a hole, the first thing to do is stop digging. Our work suggests that MCL can at the very least give machines the same simple wisdom, which would have been quite useful to the DARPA Grand Challenge robot that continued to push against a fence it could not see, burning out its motors; or to the NASA satellite that, having obeyed instructions to turn away to point at another part of the sky, was thus rendered unable to receive further instructions and so stayed in that useless position until natural orbital forces re-oriented it.

Noticing that there is a difference between what actually happened and what was expected to happen—that something has gone wrong—is the key step required to get MCL started. So, central to our approach is building systems that don't simply do things, but *know* what they are supposed to be achieving. That is, they should have specific expectations for their own performance and the outcomes of their actions. Knowing may be half the battle, but it is only half; the next steps are also crucial. Rather than simply recognize problems, report them, and wait for human system designers to fix them, MCL systems are also self-diagnosing and self-repairing.

[1] This intuitive characterization of human problem solving is also supported by work in cognitive psychology. See [16, 13, 14] for more information.

Repairs can be as simple as trying a failed action again; or trying a new plan for the same goal, dynamically learning new operators; or using trial and error; or taking advice from a human; or embarking on self-training; and so on.

It is perhaps this aspect of our approach that is most distinct from the relevant prior work on which we are building, including case-based reasoning in general, and introspective reasoning and learning from expectation failure in particular [9, 10, 11, 18]. For instance, MCL systems don't just use expectation failures to trigger a single learning strategy such as the refinement of planning operators, but can choose this option from among several; nor are MCL systems limited to selecting among multiple *learning* strategies (as with [6]), but can implement any of multiple disparate options for addressing anomolies. In addition, MCL systems don't just wait for a response plan to complete and then re-measure performance on the original task; rather they monitor the response just as they would any other action taken by the system, and can intervene if the repair itself appears to be failing. Finally, the MCL systems we envision don't just work in a single domain with specialized knowledge, but across many different domains, using ontologies capturing general coping strategies.

Imagining such systems is quite different from building them. To test the viability and domain generality of this approach, we have added MCL components to a number of otherwise self-ignorant systems. This work has convinced us that MCL is indeed a powerful approach for endowing automated systems with the flexibility needed to deal with surprise. While these implementations are domain-dependent and differ significantly in various respects, they share some common features that serve to characterize the key aspects of MCL.[2]

We can briefly indicate some of these common features. Noting an anomaly amounts to noting a mismatch between an expectation Exp(E) and an observed outcome Obs(¬E). This is the basis for the Note stage in all our implementations and domains.[3] The Assess stage identifies the anomalous E and ¬E (in some current context C) as being of some type T: Type(E,¬E,C,T). Here T might turn out to be highly domain specific (e.g., sensor error, if the domain and context are sufficient to determine that), or very general (e.g., simple logical contradiction). In addition, the Assess stage has, for each anomaly-type T, a prioritized list of possible responses. Finally, the Guide stage enacts responses from that list; if a given response fails, another is selected. Response failure can also lead to recursion if the NAG cycle is triggered by an anomaly in the attempt to guide a response. There is the danger here of non-terminating recursion, with repairs applied to failed repairs ad infinitum. However, as these repairs to repairs accumulate, the expected time to completion of the original task will move farther into the future, forcing MCL ultimately to fall back on a response such as asking for help (which may be costly but powerful) or simply giving up and moving on to something else. One final point: expectations themselves can be learned, as well as modified, by experience, and this is one of the most powerful aspects of MCL. Thus, not only is a particular instance of an anomaly dealt with by an MCL-endowed system, but the system's future expectations may change as a result. MCL can deal with learned expectations that are unreliable by either gathering more data to further refine them or by simply abandoning them altogether.

We have implemented these ideas in highly varied domains, including natural-language human-computer dialog, robot navigation, nonmonotonic reasoning, reinforcement learning, and a tank game. Three of these systems are described below, after which we will describe our efforts to implement domain-general MCL.

2.1 MCL-Enhanced Reinforcement Learning

Reinforcement learning (RL) is an established methodology that works very well in many settings, notably ones in which the reward structure is static or nearly static. But when that structure is changed suddenly and significantly, the performance of RL degrades severely and recovers excruciatingly slowly. In essence, RL algorithms need to "unlearn" what they have learned, step by step, since they have no way to recognize that the reward structure has changed, let alone assess what can be done about it. Yet it is clear that, given a drastic change that makes previous learning useless, the best policy is simply to throw it out and start over.

Using a variety of reinforcement learning algorithms (Q-learning, SARSA, and prioritized sweeping) we experimented with a simple 8×8 grid world with rewards in cells $(1, 1)$ and $(8, 8)$. The learner

[2] For a more in-depth look at MCL and the motivation behind it, see [4].

[3] In the case of logic-based domains, an anomaly often takes the form of a direct contradiction, E and ¬E. This is the case, for instance, not only in the nonmonotonic reasoning domain, but also in the natural-language domain discussed below. For these, we employ active logic [7], a time-sensitive inference engine specifically designed to allow an automated agent to reason in real time about its own ongoing reasoning, noting direct contradictions rather than inadvertently using them to derive all sentences.

was trained for 10,000 steps, then the rewards were switched, and learning continued for another 10,000 steps. We compared the performance of standard RL algorithms to MCL-enhanced versions of the same algorithms. The MCL-enhanced RL algorithms maintained and monitored expectations about such things as average reward per step, value of future rewards, and average time to next reward. When these expectations were violated, the enhanced algorithms assessed the nature of the violation and, using a simple decision tree, chose one of the available repairs (which included: ignoring the problem, adjusting the learning parameter, or throwing out the current action policy and starting over).

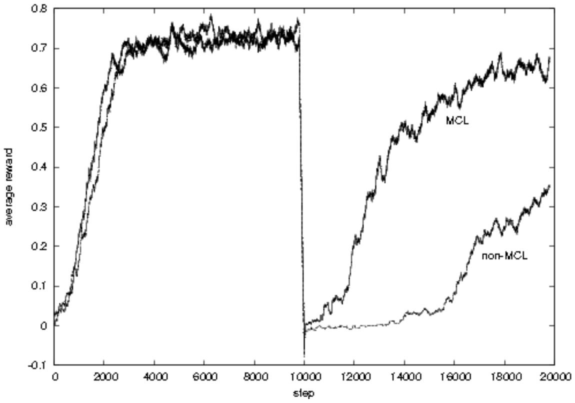

Figure 1: Learning curves for a standard Q-learner versus an MCL-enhanced Q-learner in the face of changes to the reward structure.

A typical result is shown in Figure 1. The horizontal axis is the step number, and the vertical axis is average reward. Performance rises sharply and levels off until step 10,000 when the reward-switching occurs. At that point, performance falls dramatically and then begins to recover. However, the standard RL algorithms (in this case, Q-learning, seen as the lower curve) recover far more slowly and far less completely than the MCL-enhanced versions (the higher curve) of the same algorithms. Though given enough experience both variants will asymptote to the same point, in our experiments we found that the greater the degree of change in reward (such as swapping rewards for penalties, and vice versa), the greater and longer lasting were the transient benefits of MCL [3].

2.2 MCL-Enhanced Human-Computer Dialog

Another application area for MCL is natural language human-computer dialog. Natural language is complex and ambiguous, and therefore, communication always contains an element of uncertainty. To manage this uncertainty, human dialog partners continually monitor the conversation, their own comprehension, and the apparent comprehension of their interlocutor. Human partners elicit and provide feedback as the conversation continues, and make conversational adjustments as necessary. We contend that the ability to engage in this meta-dialog is the source of much of the flexibility displayed by humans when they engage in conversation [17]. We have demonstrated that enhancing existing dialog systems with a version of MCL that allows for meta-dialogic exchanges improves performance.

In one specific case tested, a user of TRAINS-96 [1], a simulation of a national train network that is controlled by natural language, tells the system to "Send the Boston train to New York." If there is more than one train in Boston, the system may well choose the wrong one to send—the user may have in mind the train that runs regularly between Boston and New York and so might respond: "No, send the *Boston* train to New York!" Whereas the original TRAINS-96 dialog system responds to this apparently contradictory sequence of commands (Send, Don't send, Send) by once again sending the very same train, our MCL-enhanced version of TRAINS notes the anomaly (i.e., the contradiction in commands) and, by assessing the problem, identifies a possible explanation in its choice of referent for "the Boston train". The enhanced system then chooses a different train the second time around, or if there are no other trains in Boston, it will ask the user to specify the train by name. The details of the implementation, as well as a specific account of the reasoning required for each of these steps, can be found in [20].

More recently we have built another dialog system, ALFRED[4], that uses the MCL approach to resolve a broader class of dialog anomalies. The system establishes and monitors a set of dialog expectations related to time, content and feedback. For example, if the user says "Send the Metro to Boston", ALFRED notices that it doesn't know the word 'Metro' (a failure of the expectation that it will find input words in its dictionary). Alfred's first response is to try to determine what it can about the unknown word. Since Alfred knows the command "send" and its possible arguments, it is able to determine that "Metro" is a train. If it cannot determine from this which train the user is referring to, it will request specific help from the user, saying: "Which train is 'Metro'?" Once the user tells the system that 'Metro' is another word for 'Metroliner', it is able to correctly implement the

[4] **A**ctive **L**ogic **F**or **R**eason **E**nhanced **D**ialog

user's request [12].

2.3 A Bolo Player

Bolo is a multi-player tank game which takes place in a world that contains various terrain types (roads, swamps, walls, etc.), refueling bases, and pillboxes. There are three types of pillboxes: neutral pillboxes fire on all tanks, friendly pillboxes fire only on other players' tanks. and dead pillboxes pose no threat and can be captured to make them friendly. An important strategy in Bolo is to capture pillboxes, make them friendly, and then use them either offensively or defensively. Figure 2 shows a Bolo tank approaching a neutral pillbox.

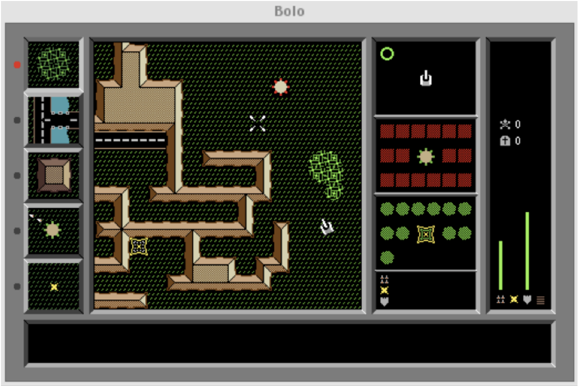

Figure 2: A Bolo tank approaching a neutral pillbox beyond the trees with a refueling station behind the walls to the left of the tank.

Bolo can be played by humans, but it can also be played by programs. Such artificial Bolo players tend to play quite poorly and are easily fooled when unexpected complications arise (change of terrain, more dangerous pillboxes, etc). Thus Bolo provides a good challenge domain in which to test MCL.

Our MCL-enhanced Bolo player is controlled by a simple Hierarchical Task Network (HTN) planner with primitive actions that ground out in controllers. It maintains a variety of expectations, the primary one being that the tank it controls will not be destroyed.[5] The initial HTN allowed the player to locate and capture dead pillboxes. However, the player did not have a plan to deal with hostile pillboxes, which fire on the tank, and so it was destroyed in its first such encounter. At this stage—when an expectation fails—MCL has a suite of actions to choose from, including means-ends analysis and operator refinement [10, 22]. In one scenario, our MCL-enhanced Bolo player was able to discover that firing on pillboxes offered a solution to the problem, even though it had no previous knowledge of the effect of that particular action. More precisely, the MCL component searched through its past experience to try to locate salient differences in the conditions under which it succeeded in taking pillboxes, and those in which it failed. It found that only pillboxes with intact armor destroyed the tank, so the next step was to see if it had any actions that could reduce the armor of a pillbox. If it had known about an action that would do that, it would have tried the action immediately. In the case we tested, it had no such knowledge. Thus, it used a heuristic to rank all its actions according to how likely they were to have the desired effect, and then tested them until it found one that worked. Note how MCL turns failure into opportunity: in each case the system learned more about what effects its actions did and did not have, and in a way organized to support its ongoing mission.

3 Toward a Single Domain-Independent MCL Implementation

Humans are good at dealing with surprise (employing the NAG cycle) not only in a few specific domains, but generally. This, together with the successes we have seen with executing MCL in highly varied domains, suggests to us that a single, generalized MCL implementation might also be able to negotiate surprise across domains.

The idea of a domain-general MCL is this: there can be a general-purpose MCL-based implementation such that any given automated host system S that (as is typical) exhibits considerable brittleness can be interfaced with that implementation, allowing the latter to monitor and (when needed) guide S into significantly improved performance with respect to unanticipated situations, without the usual human recoding.

To achieve this goal, we are now building such a general purpose MCL that encodes domain-general knowledge about anomalies in three ontologies, as shown in Figure 3. There is one ontology for each phase of the NAG cycle.[6]

The note phase of the NAG cycle uses an ontology of *indications*, which are sensory or contextual

[5] When the tank is destroyed, it reappears at a random location on the map.

[6] Although the ontology-based view of MCL described here is still in the process of being implemented, it cleanly captures the more ad hoc implementations of the various MCL-enabled systems that we have developed.

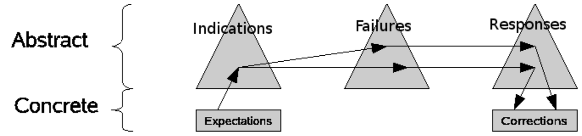

Figure 3: The three MCL ontologies: Indications, Failures and Repairs

cues that something is wrong. Within this ontology, host-specific nodes at the lowest levels encode expectations about the behavior of sensor, state, and other values. Violations of these expectations activate domain-general nodes that characterize anomalies. For example, a reinforcement learning (RL) host system may maintain expectations about receiving rewards in certain states which, when violated, will trigger a domain-general indication such as MISSING DATUM.

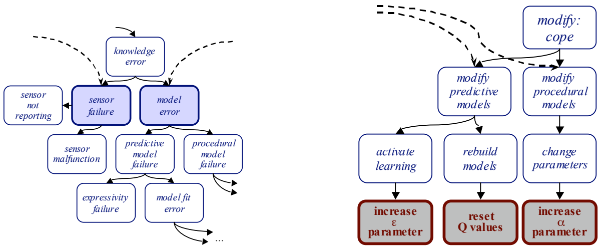

Figure 4: Fragments of MCL's failure and response ontologies which are used by the assess and guide phases of the NAG cycle, respectively.

Figure 4 shows fragments of the MCL *failure* (on the left) and *response* (on the right) ontologies, which are used by the assess and guide phases of the NAG cycle, respectively. In the former, dashed arrows are inter-ontological links coming from the indications ontology. That is, indications are linked to the types of failures they suggest. To continue our example of an RL host system, a MISSING DATUM could be due to a SENSOR FAILURE or a MODEL ERROR (i.e., the expectation itself is wrong). These failure types can be generalized or specialized by moving up or down links within the failure ontology.

Failure nodes are linked to candidate courses of action in the response ontology, again denoted with dashed arrows. For example, if there is in fact a MODEL ERROR, and the model in question is a predictive model, one possible response is to MODIFY PREDICTIVE MODELS. This response is abstract, so it must be made concrete in terms of the host system by following downward links in the ontology. In the case of an RL host based on a Q-learning algorithm, a Q-table represents expectations about future rewards, and can be adapted by increasing exploration, increasing the learning rate, or, if all else fails, reseting the Q-table and starting from scratch. The host specific response node (the shaded nodes in the response ontology in the figure) with the highest expected utility is chosen and implemented.

4 Related Work

As has been noted already, the brittleness problem is not new, and autonomous systems have always needed mechanisms to deal with the unexpected. Systems as early as Shakey [8] were able to re-plan in light of failures or unmet preconditions. In recent years there has been a great deal of work in such areas as reactive computing, machine learning, planning and re-planning, and fault detection, isolation and recovery (FDIR); see [15, 23, 21, 19] for just a brief sampling, and [2] for a general overview of metacognitive approaches to reasoning and learning.

One thing that is distinctive about our approach is its insistence on the general applicability of the NAG cycle, along with an architecture that encapsulates the metacognitive components, engaging them only when an expectation has been violated. This has advantages over some approaches—for instance, many reactive/adaptive systems (such as reinforcement learners) deal with change by continually training. This requires continual exploration, or deviation from the optimal action policy, whereas MCL systems can act optimally until they notice that something is wrong and then take remedial actions focused on the problem at hand.

For instance, although FDIR is clearly motivated by the same concerns as MCL—and has the same tripartite structure as the NAG cycle: (FD), (I), (R)—it typically takes an expert-systems or model-based approach to the issue that imposes significant limitations on the types and range of available diagnoses and repairs, often limiting the options to specific solutions known in advance to fix specific problems. By including the possibility of generating novel hypotheses about the causes of its performance, and learning new models, operators or action policies to deal with failures, MCL greatly increases the range of possible responses the system could implement. This is in line with our general preference (which we admit may not be appropriate for every domain) for increasing the agency and autonomy of our systems, and especially their freedom of action in responding to problems, rather than limiting it and hoping that circumstances do not stray from the anticipations of the system designer. Again, this increased freedom is highly focused, coming into play only when anomalies are noted that directly impact the performance of the system, about which expectations are maintained and monitored. MCL systems are self-aware, self-guided learners, able to decide whether, what, when, and how to learn; this combination of reasoning and learning, in which each can guide the other under the direction of the system itself, is one of the distinctive elements of the MCL approach.

5 Conclusion

Humans confront the unexpected every day, deal with it, and often learn from it. AI agents, on the other hand, are typically brittle—they tend to break down as soon as something happens that their creators didn't plan for. Brittleness may be the single most important and most difficult problem facing AI research. It is difficult because it is impossible to prepare an AI system for every possible contingency. It is important not only because brittle systems become unproductive when they fail, but also because if the goal is human level intelligence, then systems must exhibit human-like flexibility.

We believe that it is possible to replicate human-like behavior by furnishing AI systems with one of the methods that humans use to deal with the unexpected, namely the Note-Assess-Guide cycle. In order to do this, we must enable AI systems to help themselves; they must establish expectations and monitor them, note failed expectations, assess their causes, and then choose appropriate responses.

We have, in fact, implemented this method in several distinct types of systems: a reinforcement learner, a human-computer dialog agent, a tank game, a robot navigation system and a commonsense (non-monotonic) reasoner. In each case, the performance of the system was enhanced by MCL mechanisms. This, we believe, is promising enough to warrant our next step: domain-general MCL. Our vision of this single, domain-independent implementation of MCL is that it can be interfaced with any brittle automated system and thereby improve its performance in unexpected situations.

References

[1] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. Robust understanding in a dialogue system. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics*, pages 62–70, 1996.

[2] Michael L. Anderson and Tim Oates. A review of recent research in metareasoning and metalearning. *AI Magazine*, 28(1):12–16, 2007.

[3] Michael L. Anderson, Tim Oates, Waiyian Chong, and Don Perlis. The metacognitive loop: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *Journal of Experimental and Theoretical Artificial Intelligence*, 2006.

[4] Michael L. Anderson and Donald R. Perlis. Logic, self-awareness and self-improvement: The metac ognitive loop and the problem of brittleness. *Journal of Logic and Computation*, 15(1), 2005.

[5] Ronald J. Brachman. (AA)AI, More Than the Sum of its Parts. *AI Magazine*, 27(4):AAAI Presidential Address, Winter 2006.

[6] Michael T. Cox and Ashwin Ram. Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112(1-2):1–55, 1999.

[7] J. Elgot-Drapkin and Don Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:75–98, 2006.

[8] Richard E. Fikes. Monitored execution of robot plans produced by STRIPS. Technical Report 55, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Apr 1971.

[9] Susan Fox and David B. Leake. Using introspective reasoning to refine indexing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 391–399, 1995.

[10] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 87–95, 1994.

[11] Kristian J. Hammond. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 556–560, 1986.

[12] Darsana P. Josyula. *A Unified Theory of Acting and Agency for a Universal Interfacing Agent*. PhD thesis, Department of Computer Science, University of Maryland, College Park, 2005.

[13] Howard H. Kendler and Tracy S. Kendler. Vertical and horizontal processes in problem solving. *Psychological Review*, 69:1–16, 1962.

[14] Howard H. Kendler and Tracy S. Kendler. Reversal-shift behavior: Some basic issues. *Psychological Bulletin*, 72:229–232, 1969.

[15] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 1051–1058, 1990.

[16] Thomas O. Nelson, John Dunlosky, Aurora Graf, and Louis Narens. Utilization of metacognitive judgments in the allocation of study durying multitrial learning. *Psychological Science*, 5:207–213, 1994.

[17] Don Perlis, Khemdut Purang, and Carl Andersen. Conversational adequacy: mistakes are the essence. *International Journal of Human-Computer Studies*, 48:553–575, 1998.

[18] Roger C. Schank and David B. Leake. Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40(1-3):353–385, 1989.

[19] Eleni Stroulia and Ashok K. Goel. Redesigning a problem-solver's operations to improve solution quality. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 562–567, 1997.

[20] David R. Traum, Carl F. Andersen, Waiyian Chong, Darsana Josyula, Yoshi Okamoto, Khemdut Purang, Michael O'Donovan-Anderson, and Don Perlis. Representations of dialogue state for domain and task independent meta-dialogue. *Electronic Transactions on Artificial Intelligence*, 3:125–152, 1999.

[21] Roman van der Krogt, Mathijs de Weerdt, and Cees Witteveen. A resource based framework for planning and replanning. *Web Intelligence and Agent Systems*, 1(3-4):173–186, 2003.

[22] Xuemei Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[23] Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 971–978, 1996.