Metareasoning

Thinking about Thinking

edited by Michael T. Cox and Anita Raja

foreword by Eric Horvitz

The MIT Press
Cambridge, Massachusetts
London, England

2011

Portinale (Eds.),
Lecture Notes in

E *Transactions on*

truction of learn-

ificial Intelligence,

to improve case-

iting estimates of
cci (Eds.), *Proceed-*
Lecture Notes in

case-based reason-

tion by introspec-
First International
. Intelligence (pp.

f-models. In M. T.
AAI Workshop (pp.

ine the case-based
(3), 319–339.

. (2005). Retrieval,
5–240.

rovement through
20(1), 1–36.

reduce the quality
), 181–192.

12 The Metacognitive Loop and Reasoning about Anomalies

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

Murphy's Law states, "if anything can go wrong, it will." Though it is more of an adage than a law, it is surprisingly predictive. For each of the fifteen participants in the 2004 DARPA Grand Challenge driverless car competition, Murphy's Law held true. Each of the entries was an impressive engineering feat; to receive an invitation each team's vehicle had to navigate a mile-long preliminary obstacle course. Yet in the longer course, every one of the driverless vehicles encountered a situation for which it was unprepared: some experienced mechanical failures, while others wandered off course and into an obstacle their programming could not surmount (Hooper, 2004).

The DARPA Grand Challenge highlights the enormity of the defensive design task, in which the engineer must attempt to enumerate the ways in which a system might fail so that they can be appropriately managed. For sophisticated computer systems, particularly autonomous systems operating in the real world, this is a great challenge. In general, no system designer, whether we are talking about a learning system, a planning system, or any other artificial intelligence (AI) technology, can enumerate all the possible contingencies his or her system will encounter. This is not a unique observation. Such a view has been pointed out before (Brachman, 2006). Systems that learn and adapt attempt to address this, but learning processes themselves are also constrained to work in the space for which they were designed. Learning systems can improve robustness, but only in the situations for which they are designed.

Consider, though, a driverless vehicle that has become stuck on an embankment (a fate of several of the participants in the grand challenge). If that vehicle had a self-model that allowed it to reason about its own control and sensing capabilities, it may have been in a position to notice and diagnose its own failure. Such a system would also have the ability to reason about which of its cognitive components, whether they be controllers, learning algorithms, or planners, might allow the system to recover from the current failure, or at least prevent it from happening the next time. How can an AI system create such a self-model so that it can diagnose its own failures?

We propose that at some level of abstraction, the ways in which a system can fail are finite. Thus, a domain-general metareasoning component can be developed and equipped with knowledge of how systems fail (and how to recover from these failures). This component, when integrated with an existing AI system (which we will call the *host*), will allow that system to diagnose failures and thus become more robust.

In this chapter we present our architecture for generalized metacognition aimed at making AI systems more robust. The key to this enhancement is to characterize a system by its expectations each time it engages in activity, to watch for violations of system expectations, and to attempt to reason in an application-general way about the violation to arrive at a diagnosis and plan for recovery. Our architecture is called the *metacognitive loop* (MCL), and we present it here along with details of its implementation.

The Metacognitive Loop

Human intelligence manages to work not just in everyday situations, but also in novel situations, and even in significantly perturbed situations. For our purposes, we define a perturbation as a change in conditions under which an agent (human or artificial) has obtained competency.

Suppose someone who has spent his entire life in the desert is suddenly dropped in the middle of a skating rink. This person has learned to walk, but never on ice. His usual gait will not produce the desired result. In coping with this new situation, he starts by noticing that the proprioceptive feedback he is receiving is unusual in the context of walking. He must become more aware of what he is doing and reason sensibly about the situation. This allows him to assess what has changed or gone awry. Once he has made an assessment, he must respond to the perturbation by modifying his usual behavior: become more cautious and deliberate, or attempt to learn the dynamics of walking on ice.

Dealing with perturbations invariably involves reasoning about one's own self: about one's abilities, expectations, and adaptivity. We recognize when we possess a necessary capacity or whether we need to acquire it. What would be required of a computer system that endeavored to have that same level of robustness?

An AI system capable of reasoning about its own (reasoning) capabilities is said to possess the ability of metareasoning. A typical metareasoner can be laid out as in figure 12.1, consisting of a sensorimotor subsystem, shown in the figure as the ground level and responsible for sensing and effecting changes in an environment; a reasoning subsystem, shown as the object level and responsible for processing sensory information and organizing actions at the ground level; and a metareasoning component, shown as the metalevel and responsible for monitoring and controlling the application of components at the object level (see Cox and Raja, this vol., chap. 1).

We are developing an embedded, general-purpose metareasoner based on this basic architecture. The metacognitive loop (MCL) is a metalevel component that endows

ı these failures).
we will call the
ɔre robust.
nition aimed at
ɔ characterize a
:or violations of
ɩeral way about
tecture is called
ı details of its

ɩut also in novel
ɔoses, we define
ɩan or artificial)

ddenly dropped
ɩever on ice. His
ːw situation, he
unusual in the
ɩing and reason
ːd or gone awry.
n by modifying
pt to learn the

one's own self:
ːn we possess a
ɔe required of a
ess?
ɔilities is said to
l out as in figure
he ground level
ːnt; a reasoning
ensory informa-
ng component,
ing the applica-
hap. 1).
ɟed on this basic
ːnt that endows

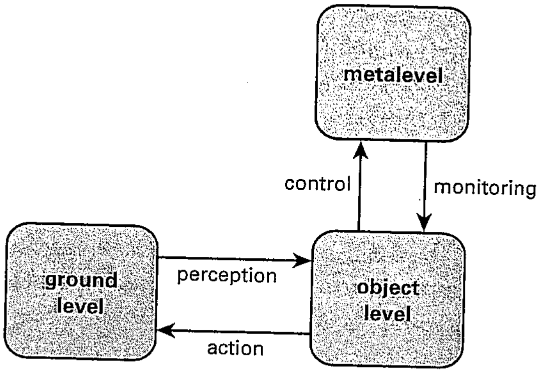

Figure 12.1
An overview of a typical metareasoning system.

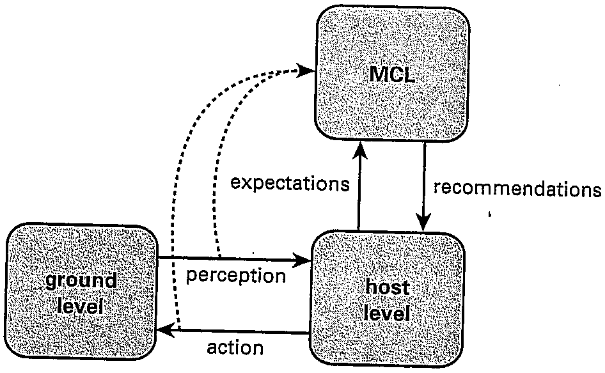

Figure 12.2
An overview of an MCL-enhanced AI system.

AI systems with self-modeling, monitoring, and repair capabilities. An overview of an MCL-enhanced system can be seen in figure 12.2. A reasoning system that employs MCL (called the *host system*) makes explicit its components, capabilities, actions, percepts, and internal state information to compile the infrastructure necessary for a self-model. Additionally, the host declares expectations about how its activities will affect the perceptual and state information. MCL monitors the operation of the host (including its actions and sensory feedback) against its expectations, waiting for violations to occur. When a violation of expectations is detected, it employs a combination of a domain-general problem solver and the host's self-model to make recommendations on how to devote computational resources to anomalous host behavior.

The operation of MCL is analogous to the thought process of the human walking on ice presented above. It can be thought of as a background process consisting of three steps: (i) monitoring for and noticing anomalies; (ii) assessing them (probable causes, or severity); and (iii) guiding an appropriate response into place.

The monitoring phase corresponds to an agent's "self-awareness." As an agent accumulates experience with its own actions, it develops expectations about how they will unfold. An agent might expect an internal state to change to a new value, for a sensor to increase at some rate, or for an action to achieve a goal before some deadline. As the agent engages in a familiar behavior, deviations from expectations (anomalies) cause surprise, and initiate the assessment phase.

In the assessment stage of MCL, a profile of the anomaly is generated. How severe is the anomaly? Must it be dealt with immediately? What is its likely cause? This anomaly profile enables MCL to move on to the guide state, where a response will be selected to either help the agent recover from the failure, prevent it from happening in the future, or both. Once this response is guided into place by the host system, MCL can continue to monitor the situation to determine whether or not the response has succeeded. Should MCL determine that its initial response has failed, it can move down its list of possible responses until it succeeds, decides to ask for help, or move on to work on something else.

Domain-General MCL

Implementing our MCL-enhanced pilot applications has provided two key insights into building robust AI systems. First, building systems that employ an MCL component requires a structured understanding of how the system and all of its parts function. Object-level capabilities and expected behaviors must be known or learnable such that the metareasoner can detect any perturbations to the system. Indeed, in similar work great attention is paid to the methodologies that enable self-modeling and robust behavior in AI (Stroulia, 1994; Ulam, Goel, Jones, & Murdoch, 2005; Williams & Nayak, 1996), and in the literature of fault detection, isolation, and recovery (FDIR) (Frank, 1990; Isermann, 1997).

The second insight is that although there may be many different perturbations possible in a given domain, there are a limited number of distinct ways in which they may create system failures, and generally an even smaller number of coping strategies. Can we produce a taxonomy of the ways in which AI systems fail, and reason about failures using the general concepts present in that taxonomy, such that one general-purpose reasoner can be useful to a wide variety of host systems and domains? Indeed, our primary scientific hypothesis is that the answer to this question is "yes," and our current research seeks to determine to what extent this hypothesis is correct.

It is useful to consider two different forms of generalized utility here. A system/domain-general MCL would be coupled "out-of-the-box" with any of a wide variety of host systems and in a wide variety of domains; the host would at a minimum need only provide MCL with expectations and monitoring information and specify any tunable actions it might have. An anomaly-general MCL would have a sufficiently high-level typology of anomalies such that virtually all specific anomalies would fall into one type or another. Since actual instances of anomalies tend to be system or domain specific, the two dimensions are not totally independent. However, a system/domain-general MCL would have a protocol design facilitating a kind of "plug and play" symbiotic hook-up, where the system/host need only provide and receive data from MCL in a specified format, even if MCL might not be equipped to handle anomalies in some domains. An anomaly-general MCL, by contrast, would be equipped to process virtually any anomaly for any system or domain, even if it might be tedious to provide the add-on interface between them. Combining the two gives the best of both worlds: easy hook-up to any host (as long as the designer follows the communication protocol) and an ability to deal flexibly with whatever comes its way. Indeed, the primary difference between MCL and much of the related work, perhaps best exemplified by that of Goel, is that rather than requiring a complete self-model, MCL can operate with more modest knowledge about expectations, the failures (probabilistically) indicated by violations thereof, and potentially effective repairs.

The current generation of MCL implements such a generalized taxonomy and uses it to reason through anomalies that a host system experiences. MCL breaks the universe of failures down into three ontologies that describe different aspects of anomalies, how they manifest in AI agents, and their prescribed coping mechanisms. The core of these ontologies contains abstract and domain-general concepts. When an actual perturbation is detected in the host, MCL attempts to map it into the MCL core so that it may reason about it abstractly. Nodes in the ontologies are linked, expressing relationships between the concepts they represent. The linkage both within the ontologies and between them provides the basis that MCL uses to reason about failures.

Although the hierarchical network structure of the ontologies lends itself to any of a number of graph-based algorithms, our implementation represents the ontologies as a Bayesian network. This allows us to express beliefs about individual concepts within the ontologies by probability values, to model the influence that the belief in one concept has on the others, and to use any of the many Bayesian inference algorithms to update beliefs across the ontologies as new observations are made by MCL. The core of our implementation is based on the SMILE reasoning engine.[1]

[1] The SMILE engine for graphical probabilistic modeling, contributed to the community by the Decision Systems Laboratory, University of Pittsburgh (http://dsl.sis.pitt.edu).

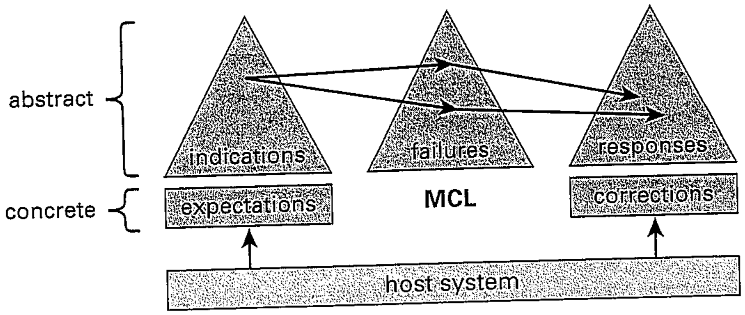

Figure 12.3
An overview of the MCL ontologies.

Each of the three phases of MCL (note, assess, guide) employs one of the ontologies to do its work (Schmill, Josyula, Anderson, Wilson, Oates, Perlis, Wright, & Fults, 2007). A flow diagram is shown in figure 12.3. The note phase uses an ontology of indications. An indication is a sensory or contextual cue that the system has been perturbed. Processing in the indication ontology allows the assess phase to hypothesize underlying causes by reasoning over its failure ontology. This ontology contains nodes that describe the general ways in which a system might fail. Finally, when failure types for an indication have been hypothesized, the guide phase maps that information to its own response ontology. This ontology encodes the means available to a host for dealing with failures at various levels of abstraction. Through these three phases, reasoning starts at the concrete, domain-specific level of expectations, becomes more abstract as MCL moves to the concept of a system failure, and then becomes more concrete again as it must realize an actionable response based on the hypothesized failure.

In the following sections, we will describe in greater detail how the three ontologies are organized and how MCL gets from expectation violations to responses that can be executed by the host system, using the MCL-enhanced reinforcement learning system as an example. To help illustrate the functions of the ontologies, we will use a previous study of ours as an example (Anderson, Oates, Chong, & Perlis, 2006). In this study we deployed MCL in a standard reinforcement learner. There, learned reward functions in a simple 8 × 8 grid world formed the basis for expectations.[2] When reward conditions in the grid world were changed, MCL noted the violation and would respond in a number of ways appropriate to relearning or adapting policies in RL systems. In a variety of settings, the MCL-enhanced learner outperformed standard

[2] Q-learning (Watkins & Dayan, 1992), SARSA (Sutton & Barto, 1995), and prioritized sweeping (Moore & Atkeson, 1993) were used.

the ontologies
right, & Fults,
ın ontology of
stem has been
se to hypothe-
ology contains
Finally, when
ıase maps that
neans available
ıgh these three
ıtions, becomes
then becomes
n the hypothe-

hree ontologies
ɔonses that can
ement learning
ies, we will use
Perlis, 2006). In
There, learned
ctations.[2] When
ıtion and would
; policies in RL
ɔrmed standard

ıoritized sweeping

reinforcement learners when perturbations were made to the world's reward structure.

Indications

A fragment of the MCL indication ontology is pictured in figure 12.4. The indication ontology consists of two types of nodes separated by a horizontal line in the figure: domain-independent indication nodes above the line, and domain-specific expectation nodes below it. Indication nodes belong to the MCL core and represent general classes of sensory events and expectation types that may help MCL disambiguate anomalies when they occur. Furthermore, there are two types of indication nodes: fringe nodes and event nodes. Fringe nodes zero in on specific properties of expectations and sensors. For example, a fringe node might denote what type of sensor is being monitored: internal state, time, or reward. Event nodes synthesize information in the fringe nodes to represent specific instances of an indicator (for example, reward not received). Expectation nodes (shown below the dashed line) represent host-level expectations of how sensor, state, and other values are known to behave. Expectations are created and destroyed based on what the host system is doing and what it believes the context is. Expectations may be specified by the system designer or learned by MCL, and are linked dynamically into indication fringe nodes when they are created.

Consider the ontology fragment pictured in figure 12.4. This fragment shows three example expectations that the enhanced reinforcement learner might produce when it attempts to move into a grid cell containing a reward. First, a reward x should be

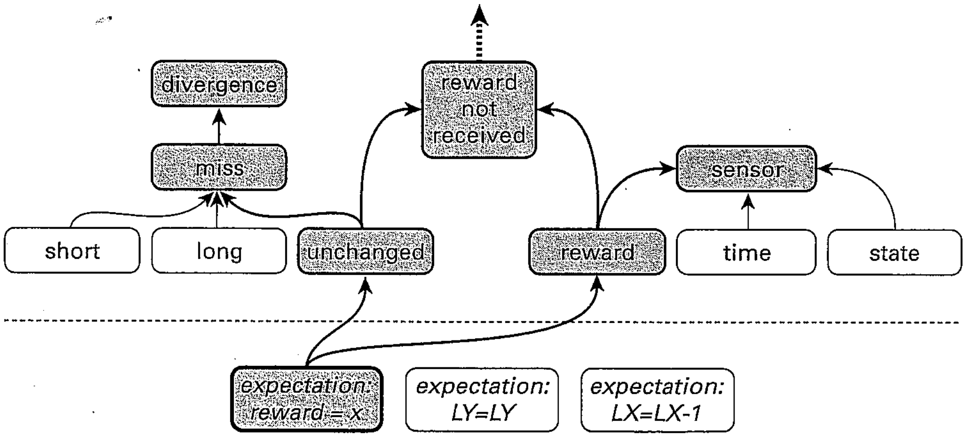

Figure 12.4
A fragment of the MCL indication ontology.

experienced at the end of the movement. Second, the sensor LY should not change. Lastly, the sensor LX should decrease by one unit.

Suppose that someone has moved the location of the reward, but LY and LX behave as if the reward were still in the original position. MCL will notice an expectation violation for the reward sensor and create a fresh copy of the three ontologies to be used as a basis for reasoning through a repair. Based on the specifics of the violation, appropriate evidence will be entered into the indication fringe to reflect the fact that a violation occurred: a change in a reward sensor was expected, but the change never occurred. The relevant expectation node in the fragment in figure 12.4 is denoted by boldface, and its influence on associated nodes in the indication ontology is denoted by heavy arrows. Through the conditional probability tables maintained by the Bayesian implementation of the ontology, MCL's belief in fringe nodes "reward" and "unchanged" will be boosted. From there, influence is propagated along abstraction links within the indication core (activating the sensor node and others). Finally, fringe-event links combine the individual beliefs of the separate fringe nodes into specifically indicated events. In figure 12.4, the "reward not received" node is believed to be more probable due to the evidence for upstream nodes. Once all violated expectations have been noted, and inference is finished, the note phase of MCL is complete.

Failures

The note stage having been completed, MCL can move to the assessment stage, in which indication events are used to hypothesize a cause of the anomaly experienced. The failure ontology serves as the basis for processing at the assessment stage.

Belief values for nodes in the failure ontology are updated based on activation in the indication ontology. Indication event nodes are linked to failure nodes via interontological links called diagnostic links. They express which classes of failures are plausible given the active indication events and the conditional probabilities associated with those relationships.

Figure 12.5 shows a fragment of the MCL failure ontology. Dashed arrows indicate diagnostic links from the indications ontology leading to the sensor failure and model error nodes, which are shaded and bold. These nodes represent the nodes directly influenced by updates in the indications ontology during the note phase in our enhanced reinforcement learning example; a "reward not received" event can be associated with either of these types of failure. The remaining links in the figure are intraontological and express specialization. For example, a sensor may fail in two ways: it may fail to report anything, or it may report faulty data. Either of these is a refinement of the sensor failure node. As such, sensor not reporting and sensor malfunction are connected to sensor failure with specialization links in the ontology to express this relationship.

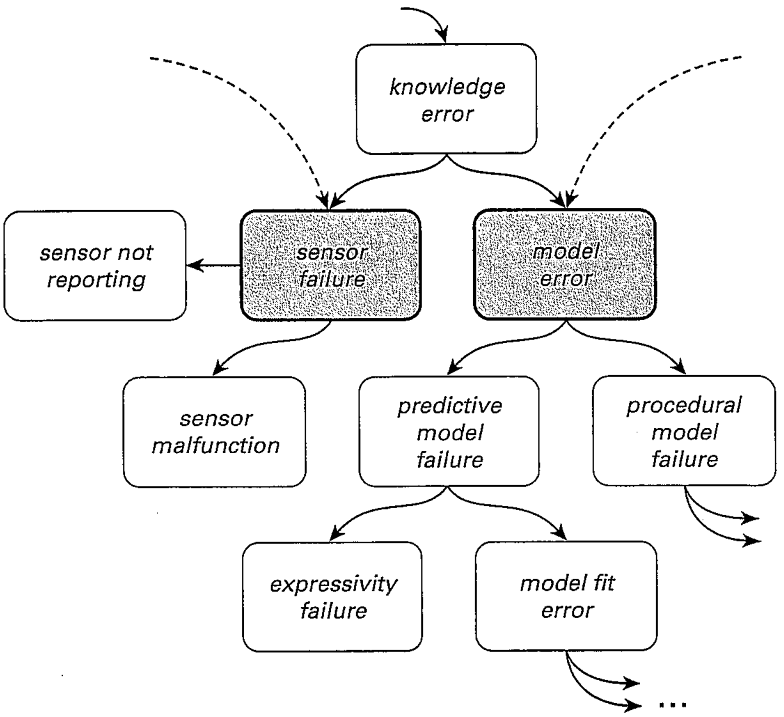

Figure 12.5
A fragment of the MCL failure ontology.

As in the note phase, influence is passed along specialization links to activate more specific nodes based on the probabilities of related abstract nodes and priors. Of particular interest in our RL example is the predictive model failure node, which follows from the model error hypothesis. The basis for action in Q-learning is the predictive model (the Q function), and failure to achieve a reward often indicates that the model is no longer a match for the domain.

Responses

Outgoing interontological links from probable failure nodes allow MCL to move into the guide phase. In the guide phase, potential responses to hypothesized failures are activated, evaluated, and implemented in reverse order of their expected cost. The expected cost for a concrete response is computed as the cost of the response multiplied by one minus the estimated probability that the response will correct the anomaly, where the cost is quantified by the host. Interontological links connecting failures to responses are called prescriptive links.

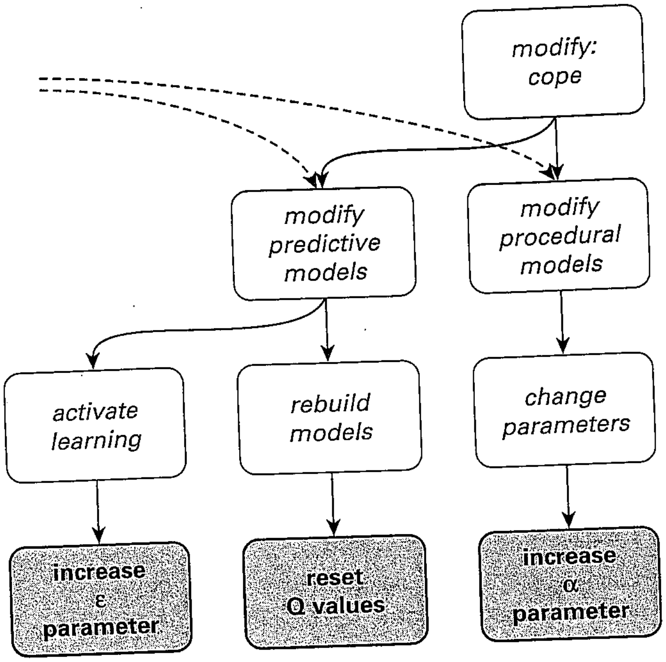

Figure 12.6
A fragment of the MCL response ontology.

Figure 12.6 shows a fragment of the MCL response ontology. Pictured are both MCL core responses (which are abstract, and shown in italics) and host-level responses (pictured in bold), which are concrete actions that can be implemented by a host system. Host system designers specify the appropriate ways in which MCL can effect changes by declaring properties (such as "employs reinforcement learning") that are incorporated into the conditional probability tables for the response nodes. Declaring "employs reinforcement learning," for example, will make nonzero the prior belief that responses, such as "reset Q values" as seen in figure 12.6, will be useful.

In the portion of the response ontology pictured, prescriptive links from the failure ontology are pictured as dashed arrows. These links allow influence to be propagated to the nodes "modify predictive models" and "modify procedural models." Like the failure ontology, internal links in the response ontology are primarily specialization links. They allow MCL to move from general response classes to more specific ones, eventually arriving at responses that are appropriate to the host. In our example, concrete nodes correspond to either parameter tweaks in Q-learning or resetting the Q function altogether.

Iterative and Interactive Repairs

Once MCL has arrived at a concrete response in the guide phase, the host system can implement that response. In our enhanced RL example, this may mean clearing the Q values and starting over, or boosting the ε parameter to increase exploration or the α parameter to accelerate learning. A hybrid system, with many components, may have several probable responses to any given indication. This is why all the activated ontology nodes are considered hypotheses with associated conditional probabilities. MCL will not always have enough information to arrive at an unambiguously correct response. MCL must verify that a response is working before it considers the case of an anomaly closed.

When a response is found to have failed, either by explicit feedback from the host, or implicitly by recurrence of expectation violations, MCL must recover its record of the original violation and reinitiate the reasoning process. The decision of when to recover a reasoning process is actually quite complex: repairs may be durative (requiring time to work), interactive (requiring feedback from the host), or stochastic. Each time an anomaly is experienced, it may be a manifestation of an all-new failure, the recurrence of a known failure, or even a failure introduced by an attempted repair. The heuristics required to make the decision of whether to initiate a new reasoning process or resume an existing one remain a topic of our ongoing research.

Once the decision has been made that a response has failed and a reasoning process should be resumed, MCL reenters and updates the ontologies in two ways. First, it revises down the belief that the "failed response" node will solve the problem, possibly driving it to zero. The inference algorithm is run and the influence of having discounted the failed response is propagated throughout the ontologies. Next, it feeds any new indications that may have occurred during the execution of the original response into the indications ontology and again executes the inference algorithm. Then utility values for concrete responses are recomputed and the next most highly rated response is chosen and recommended for implementation by the host. Once a successful response is implemented and no new expectation violations are received, the changes effected during the repair can be made permanent, and the violation is considered addressed.

Evaluation and Future Work

In this section, we describe a new system architecture we are developing that has the requisite complexity to highlight how a metareasoner can contribute to a more robust system. Through this system we also hope to demonstrate the generality of the reasoner, as MCL will have to cope with a variety of problems encountered as the various system components at the object level interact. We also include a short discussion of our planned evaluation methods.

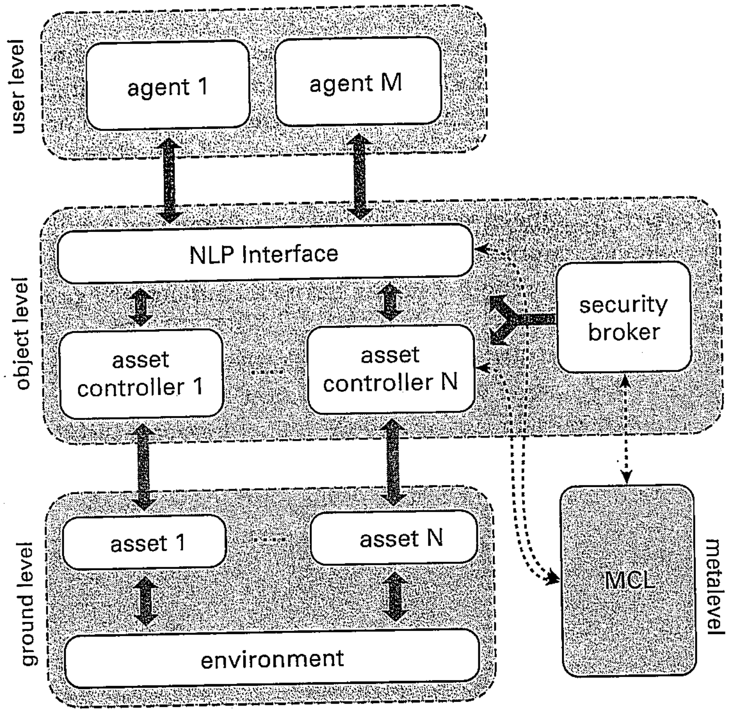

Figure 12.7
An overview of an end-to-end MCL-enhanced AI'system.

An overview of our system architecture is pictured in figure 12.7. At the ground level are "assets"—simulated agents with sensing and possibly effecting capabilities that operate in a simulated environment. The architecture is designed to be configurable; the assets might be rover units operating in a simulated Mars environment, or unmanned aerial vehicles operating in a virtual battlefield. The core of the simulation was built based on the Mars rover simulation introduced by Coddington (2007), and is currently discrete, although an obvious development path would be to transition to a two- or three-dimensional, continuous world, and eventually to actual robots acting in the real world. For the purpose of this discussion, we will use the simulated Mars rover as an example.

At the object level of the testbed system are three major cognitive components. First is the monitor and control system of each asset. It is responsible for sequencing execution of effecting and sensing actions on the actual assets. Our Mars rover controller contains a simple planner that performs route planning to navigate between waypoints on a map, while taking reasonable measures to attend to the rover's resource constraints. The rover controller can also learn operator models for the Mars environment, in a form similar to those found in STRIPS (Fikes & Nilsson, 1971).

The second object-level component is a human–computer interface that accepts natural language commands from human users. Users specify their goals to the language processor, which converts them to a goal language usable by the rover controller. The rover controller in turn generates ground-level plans to achieve the user's goals, and also manages the inevitable competition for limited asset resources.

Finally, the system contains a security broker. The security broker places constraints on both the assets and users' access to them. For example, the security broker may state that two rovers may not perform science in the same zone, or the security broker may state that user *U* may access panoramic images taken by the rover, but not specific scientific measurements in a particular zone.

The three object-level components address three distinct classes of AI problems. The rover controller is, obviously, a classic AI control problem. It requires the use of planning, scheduling, and learning, and the coordination of those capabilities to maximize the utility of the system assets. The user–asset interface is a classic AI natural language understanding, learning, and dialogue management problem. Finally, the security broker introduces security policies as a constraint, as well as information fusion.

The domain presents many possibilities for perturbations and associated system failures. Each path between the ground and object level represents a conceptual boundary, whereby one component asserts control and has expectations about the result. Consider a few possible perturbations: the human user may use unknown lexical or syntactic constructs, the user may be denied access to imagery due to conflicting security policies, or the rover may generate useless observations due to unforeseen changes in the Mars environment. Each interaction and its associated expectations will be monitored by MCL, and any violation will be mapped to the core ontologies. Possible explanations and repairs will be considered in an order consistent with prior and learned probabilities in an attempt to prevent further violations.

After building the various components outlined above, our main experimental task will be to build and test the MCL ontologies. We want to show that the MCL approach is effective and results in more perturbation-tolerant systems, and we want to show that the MCL approach is general, that the same MCL core can handle many different kinds of perturbations in many different systems.

For measuring the effectiveness of MCL, the main strategy will involve ablation studies. Performance metrics will include time to task completion and cost of task completion (e.g., fuel burned, number of messages sent), and more particularly the degree of increase in these measures as the scenario difficulty increases (Anderson, 2004). We will compare performance for three versions of the system: (1) full adaptive response: all the components/agents of the system with all parts of MCL enabled.

(2) Fixed maintenance: the components/agents of the system with no MCL, but rather a fixed maintenance policy in which available repair actions are executed in a preventative manner. For instance, the rover rebuilds models every n timesteps, recalibrates sensors every m timesteps, and so on. (3) Fixed response: The components/agents of the system with the MCL note phase working, but with the failure and response ontologies replaced by a single response, which is to run down a list of all available repairs until one succeeds. Insofar as MCL is an effective strategy for ensuring perturbation-tolerance, the performance of system version (1) should be far better than the others.

For measuring the generality of MCL, a different approach must be employed. Recall that we intend for all the MCL instantiations (e.g., for the natural language interpreter and the rover) to have identical core algorithms and core nodes (at least initially), but different fringe nodes (e.g., the type and number of expectations). In the course of development, we expect that the differences between the host systems will suggest changes to the core nodes and their connections, in order to enhance performance. The open research questions are: how much will they end up differing, and can they be reunified after optimization to generate a more truly universal MCL system? We will answer those questions in the following four steps.

First, we will allow each MCL core to be changed and trained however much is required to achieve the best enhancement over base-level performance (as noted above). After this initial training and testing, we will measure the following traits of the MCL systems: (1) which specific nodes get used in each system, and (2) what are the most frequent subtrees used in the ontologies. What we hope to see within each MCL core is that a majority of the nodes and subtrees are being activated in the course of processing the various anomalies.

What we hope to see between any two MCL cores is significant overlap between the subsets of nodes and subtrees being used in each case. Dice's coefficient is a convenient measure for this, as it allows quantification of the overlap between sets, in this case the subset of activated nodes or subtrees, as compared with the full set of available nodes or subtrees (within the MCL core) or as compared with other subsets of activated nodes of subtrees (between MCL cores). Low scores (below 0.5) would indicate poor coverage (extraneous nodes that aren't being used) or insufficient abstraction (different specialized paths being used for each different system), and would trigger a redesign of the MCL core.

Second, once we have generated MCL cores that both serve their host systems and indicate good coverage and abstraction, we will compare the whole cores to one another using a tree-edit distance measure. We hope to see that the MCL cores remain fundamentally similar (requiring few edits to turn one into the other).

Third, we will merge the two MCL cores into one, containing all the nodes and connections of the two. If the unified MCL is domain general, it should work equally well when reattached to the initial host systems. We will test this claim by rerunning the initial performance tests, and seeing if the enhanced, unified MCL works as well as the preunification specialized MCLs.

Fourth and finally, once we have a single, unified MCL core with good coverage and abstraction that works well on both host systems, we will install MCL on a system not designed by us (and with minimum modifications) that we can use as a test-host. We do this to ensure that we have not inadvertently built our original test systems so that they would automatically work with MCL.

Conclusion

We have described a generalized metacognitive layer aimed at providing robustness to autonomous systems in the face of unforeseen perturbations. The metacognitive loop encodes commonsense knowledge about how AI systems fail in the form of a Bayesian network and uses that network to reason abstractly about what to do when a system's expectations about its own actions are violated. Our aim is to provide an engineering methodology for developing metalevel interoperable AI systems and in so doing provide the benefit of adding reactive anomaly-handling using the MCL library. We have also introduced a system architecture with a number of interacting cognitive components at the object level that we believe is a useful testbed for metacognitive research.

Acknowledgments

Supported by NSF (IIS0803739), AFOSR (FA95500910144), and ONR (N000140910328).

References

Anderson, M. L. (2004). Specification of a test environment and performance measures for perturbation-tolerant cognitive agents. In R. M. Jones (Ed.), *Proceedings of the AAAI Workshop on Intelligent Agent Architectures* (pp. 11–18). Technical Report WS-04-07. Menlo Park, CA: AAAI Press.

Anderson, M. L., Oates, T., Chong, W., & Perlis, D. (2006). The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *Journal of Experimental & Theoretical Artificial Intelligence, 18*(3), 387–411.

Brachman, R. J. (2006). (AA)AI: More than the sum of its parts. *AI Magazine, 27*(4), 19–34.

Coddington, A. (2007). Motivations as a meta-level component for constraining goal generation. In A. Raja & M. T. Cox (Eds.), *Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems* (pp. 16–30). Collocated with AAMAS-07. Columbia, SC: IFAAMAS.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence, 2*(3–4), 189–208.

Frank, P. M. (1990). Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy—A survey and some new results. *Automatica, 26*(3), 459–474.

Hooper, J. (June, 2004). DARPA's debacle in the desert: Behind the scenes at the DARPA grand challenge, the 142-mile robot race that died at mile 7. *Popular Science*, 64–67.

Isermann, R. (1997). Supervision, fault-detection and fault-diagnosis methods—An introduction. *Control Engineering Practice, 5*(5), 639–652.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning, 13*(1), 103–130.

Schmill, M., Josyula, D., Anderson, M. L., Wilson, S., Oates, T., Perlis, D., Wright, D., & Fults, S. (2007). Ontologies for reasoning about failures in AI systems. In A. Raja & M. T. Cox (Eds.), *Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems* (pp. 1–15). Collocated with AAMAS-07. Columbia, SC: IFAAMAS.

Stroulia, E. (1994). *Failure-driven learning as model-based self redesign.* Doctoral dissertation, Georgia Institute of Technology, College of Computing, Atlanta.

Sutton, R. S., & Barto, A. G. (1995). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Ulam, P., Goel, A., Jones, J., & Murdoch, W. (2005). Using model-based reflection to guide reinforcement learning. In D.W. Aha, H. Muñoz-Avila, & M. van Lent (Eds.), *Proceedings of the IJCAI Workshop on Reasoning, Representation and Learning in Computer Games.* (pp. 107–112). Washington, D.C.: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3–4), 279–292.

Williams, B. C., & Nayak, P. P. (1996). A model-based approach to reactive selfconfiguring systems. In *Proceedings of the National Conference on Artificial Intelligence* (pp. 971–978). Menlo Park, CA: AAAI Press.