The Role of Metacognition in Robust AI Systems

Matthew D. Schmill and **Tim Oates**
University of Maryland Baltimore County
Baltimore, MD 21250 USA

Michael L. Anderson
Franklin & Marshall College
Lancaster, PA 17604 USA

Darsana Josyula
Bowie State University
Bowie, MD 20715 USA

Don Perlis and **Shomir Wilson** and **Scott Fults**
University of Maryland
College Park, MD 20742 USA

Abstract

As automated systems become more complex, their propensity to fail in unexpected ways increases. As humans, we often notice their failures with the same ease that we recognize our own plans going awry. Yet the systems themselves are frequently oblivious that the function they are designed to perform is no longer being performed. This is because humans have explicit expectations – about both the system's behavior and our own behaviors – that allow us to notice an unexpected event. In this paper, we propose a way for AI systems to generate expectations about their own behavior, monitor them, and attempt to diagnose the underlying failures that cause them. Once a cause has been hypothesized, attempts at recovery can be made. The process is naturally meta-cognitive in that the system must reason about its own cognitive processes to arrive at an accurate and useful response. We present an architecture called the Meta-Cognitive Loop (MCL), which attempts to tackle robustness in cognitive systems in a domain general way, as a plug-in component to decrease the brittleness of AI systems.

Introduction

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 this design paradigm, 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. 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.

Systems with self models, and the ability to reason about their own internal representations and processes possess *metacognitive* capabilities that we suggest allow them to be more robust, and simpler to implement, than the sum of their cognitive parts.

In this paper we present an 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 attempt to reason in a application-general way about the violation to arrive at a diagnosis and plan for recovery. Our architecture is called the Meta-Cognitive Loop (MCL), and we present it here, along with details of its implementation, and an example of an application of MCL to a complete end-to-end system consisting of several distinct components.

The Meta-Cognitive Loop

A system is considered brittle when unanticipated perturbations in its domain cause significant losses in performance or, worse, complete system failure. But, as noted in the discussion of the DARPA Grand Challenge above, defensive design is often a losing battle. How can the designer of an AI system hope to enumerate and plan for all possible perturbations, especially as the systems' capabilities and interactions with the real world get more and more sophisticated?

Human intelligence manages to work not just in everyday situations, but also in novel situations, and even significantly *perturbed* situations. A perturbation is defined by Merriam-Webster as "a disturbance of motion, course, arrangement, or state of equilibrium." For our purposes, a perturbation

is a change in conditions under which an agent (human or artificial) has obtained competency.

Suppose someone who has spent their 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. Their usual gait will not produce the desired result. In coping with this new situation, they start by *noticing* that the proprioceptive feedback they are receiving is unusual in the context of walking. They must become more aware of what they are doing and reason sensibly about the situation. This allows them to *assess* what has changed or gone awry. Once they have made an assessment, they must *respond* to the perturbation by modifying their usual behavior: become more cautious and deliberate, attempt to learn the dynamics of walking on ice, etc.

Dealing with perturbations invariably involves reasoning about one's own self: about one's abilities, expectations, and adaptiveness. 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 capabilities is said to possess the ability of metareasoning. A typical metareasoner can be laid out as in figure 1 (Cox and Raja 2007), 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 orginizing actions at the ground level, and a metareasoning component, shown as the *meta level* and responsible for monitoring and controlling the application of components at the object level.

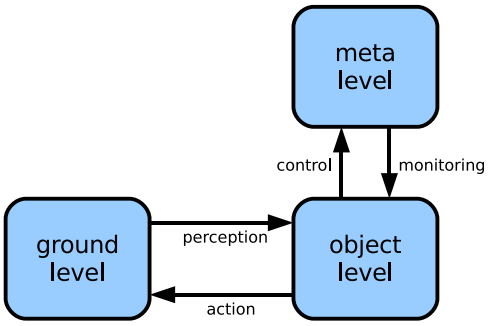

Figure 1: An overview of a typical metareasoning system.

We are developing an embedded, general-purpose metareasoner based on this basic architecture. The Meta-Cognitive Loop (MCL) is a meta-level component that endows AI systems with self-modeling, monitoring, and repair capabilities. An overview of an MCL-enhanced system can be seen in figure 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 impact the perceptual and state information it has access to. MCL monitors the operation of the host (including it's 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.

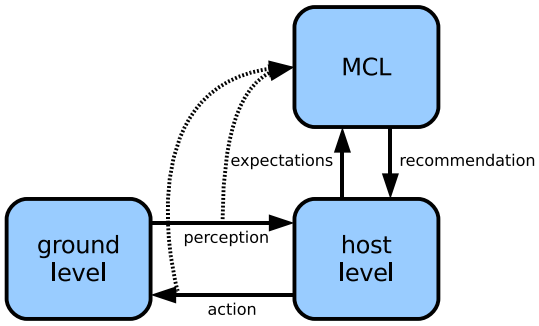

Figure 2: An overview of an MCL-enhanced AI system.

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, severity, etc.) and (iii) guiding an appropriate response into place.

The Note phase corresponds to an agent's "self-awareness". As an agent accumulates experience with its own actions, it develops expectations about how they 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 assess 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 the 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, or decides to ask for help or move on to work on something else.

In previous work, we have demonstrated the general usefulness of MCL-like metareasoning in specific applications. In one such study we deployed MCL in a standard reinforcement learner (Anderson et al. under review). There, learned reward functions in a simple 8x8 grid world formed the bases for expectations [1]. When reward conditions in

[1]Q-learning (Watkins and Dayan 1992), SARSA (Sutton and

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

In other work, we applied an MCL approach to constructing a Bolo player. Bolo[2] is a 2-dimensional multiplayer video game in which players command simulated tanks on a virtual battlefield. In this domain, we implemented a simple planner capable of sequencing simple closed-loop controllers for moving, firing, and performing other terraforming operations available in the game. Rather than devote our time to perfecting the planner, we used a simplistic approach in which plan refinement and modification was possible, and added a metareasoning component that could strategically deploy those capabilities when the simple plans failed. The combination of a simple planner and a self-aware metareasoning component provided an enhanced flexibility for the Bolo player to cope with changes in the Bolo battlefield.

Domain-General MCL

Implementing the MCL-enhanced applications discussed above has provided two key insights into building robust AI systems. First, building systems that employ a metacognitive loop embodies an engineering practice that requires a structured understanding of how their programs work. The capabilities of an automated, adaptive system must be made explicit enough for the metareasoner to know when to use them. Expectations for the behaviors executed at the object level must be known, or learnable such that the metareasoner can detect when a perturbation has occurred. Indeed, in similar work great attention is paid to the methodologies that enable self-modeling and robust behavior in AI (Ulam et al. 2005; Stroulia 1994; Williams and Nayak 1999), and in the literature of Fault Detection, Isolation, and Recovery (FDIR) (Wesley et al. 1995; Tinos and Terra 2002).

The second insight is that while 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 whether we are correct or incorrect, and to what degree.

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 ability to deal flexibly with whatever comes its way.

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 contain 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.

Though 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. [3]

Each of the three phases of MCL (note, assess, guide) employs one of the ontologies to do its work. A flow diagram is shown in figure 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: its models are out-of-date, for example. 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

Barto 1995) and Prioritized Sweeping (Moore and Atkeson 1993) were used.

[2]The game of Bolo is described in detail at http://www.lgm.com/bolo/

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

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.

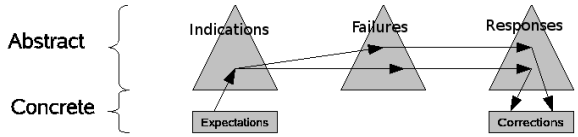

Figure 3: Overview of the MCL ontologies.

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.

Indications

A fragment of the MCL indication ontology is pictured in figure 4. The indication ontology consists of two types of nodes: domain independent *indication nodes* shown above the dashed line, and domain-specific *expectation nodes* shown below the line. 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 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 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 occured.

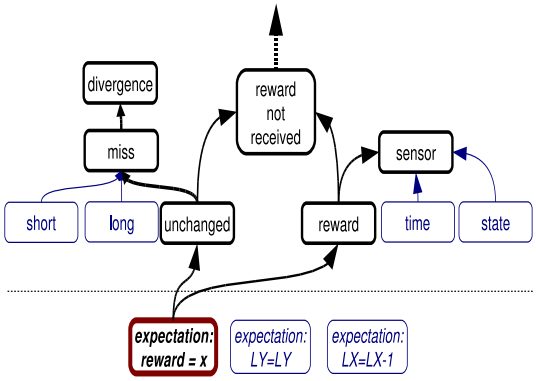

Figure 4: A fragment of the MCL indication ontology.

The relevant expectation node in the fragment in figure 4 is denoted by boldface, and its influence on associated nodes in the indication ontology are 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 seperate fringe nodes into specifically indicated events. In figure 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 assess 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 assess stage.

It is worth explaining why MCL does not map directly from indications to responses. In fact, earlier incarnations of MCL did derive responses directly from expectation violations. The failure ontology was added because of the potentially ambiguous nature of indications. In many cases, a single indication might suggest several potential failures. Similarly, a single failure might only be suspected when a subset of indications is present. The mapping between indications to failures, then, might be one-to-many or many-to-one. This rich connectivity is lost without all three ontologies.

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 5 shows a fragment of the MCL failure ontology. Dashed arrows indicate diagnostic links from the indications

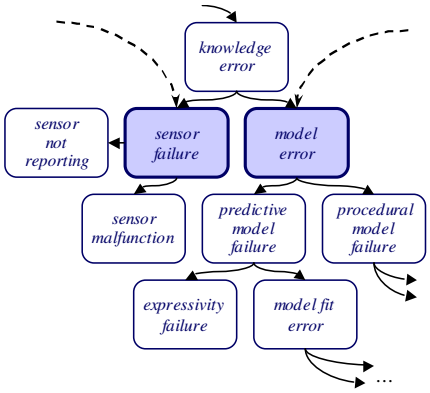

Figure 5: A fragment of the MCL failure ontology.

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.

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 probably 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. Expected cost for a concrete response is computed as the cost of the node, as declared by the host, multiplied by one minus the belief value of the node, which is taken as the probability that the response will correct the anomaly. Interontological links connecting failures to responses are called *prescriptive links*.

Figure 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 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 propogated 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.

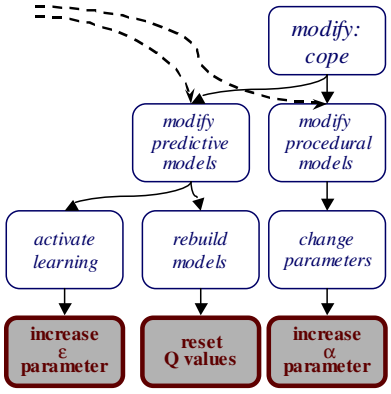

Figure 6: A fragment of the MCL response ontology.

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 learn faster. 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. Making this decision remains a topic of our ongoing research. Once the decision has been made that a response has failed, MCL *re-enters* 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 propogated 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.

An MCL Testbed

Metacognition of the type employed by the metacognitive loop is in a sense only as interesting as the collection of cognitive capabilities its host possesses. Our running example of a reinforcement learner provides relatively few sources of failure and perhaps even fewer possible responses. The reasoning performed by MCL, and the ontology nodes that come into play are consequently limited. Indeed, with so few distinct choices to be made at the object level, all of the metacognition in this example could easily be anticipated and hardcoded into the learner, blurring the lines between the object and metacognitive levels.

For the benefits of metacognition to be made clear, there must be significant choices to be made at the meta level that produce concrete benefits to overall system competency. 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 the at object level interact.

An overview of our system architecture is pictured in figure 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 unmaned aerial vehicles operating in a virtual battlefield. The core of the simulation was built based on the Mars Rover simulation introduced in (Coddington 2007), and is currently discrete, although an obvious development path would be to transition to a 2 or 3 dimensional, continuous world, and eventually 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 and 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 get at 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 dialog management problem. Finally, the security broker introduces security policies as a constraints, 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, wereby one component asserts control and has exepctations 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 unforseen 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.

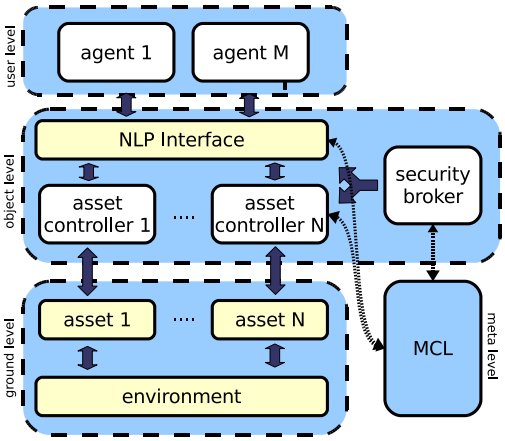

Figure 7: An overview of an end-to-end MCL-enhanced AI system.

As of the time of writing, a simulated Mars rover, and a monitor and control system as described above, is implemented in our testbed architecture. It generates simple expectations based on operator models and these expectations are monitored as it executes scripts loaded with science goals for the rover. Where the proverbial rubber hits the road, the rover has 7 actions, and a mix of 8 sensors and state variables, both continuous and discrete. We can perturb the controller through both the environment (by, for example, blocking a route between waypoints) and the rover (by, for example, causing a sensor to behave improperly).

Similarly, the NLP interface (a system called ALFRED (Anderson et al. 2004)) is undergoing revisions to interoperate with MCL. The grammatical and dialogical analysis of an utterance (the spelling, syntactic structure, the context-independent and dependent meanings, reference tracking, etc.) is coded in an active logic-based language (ALMA, see (Anderson and Perlis 2005)) and designed around a BDI architecture. These types of knowledge and reasoning representations lend themselves readily to self-monitoring and reasoning. Hence, MCL could end up playing an increasingly integral part in both the grammatical and the dialogical tasks of the NLP interface. That is, a system designed to be a general, commonsense reasoner may play essential roles in sentence level parsing, dialogue management, and the learning of both.

Conclusion

We have described a generalized metacognitive layer aimed at providing robustness to autonomous systems in the face of unforseen 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 meta-level interoperable AI systems and in so doing provide the benefit of adding reactive anomaly handling using the MCL library.

It is important to emphasize that MCL is not a kind of magic bullet that figures out and then performs perfect actions for every situation. Rather, the metacognitive layer can step back and assess a difficulty – possibly deciding that it does not have the wherewithal to resolve it, and possibly availing itself of options such as asking for help, giving up (rather than wasting time), using trial-and-error, or suggesting application of any of the adaptive behaviors implemented at the object layer, if it has the capacity to do so.

This kind of commonsense approach to anomalies tends to serve humans very well; though there is some evidence to suggest that metareasoning can be detrimental to decision-making (Wilson and Schooler 1991), we suggest that this may be true only in cases where the cognitive or object level is functioning adequately. Said differently, when the expectations of a cognitive process are violated, it is appropriate to perform diagnosis and response at the meta-level precisely because the violation is due to a failing somewhere at the object level. When the meta-level is engaged *without a congitive end*, the potential to reorganize the object level to overall detriment exists.

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. We believe it will provide a setting complex enough to demonstrate both the utility and generality of our generalized MCL ontologies, bringing together issues of information fusion, security policies, human-computer dialog, and monitor and control of autonomous systems.

References

Anderson, M. L., and Perlis, D. R. 2005. Logic, self-awareness and self-improvement: {T}he metacognitive loop and the problem of brittleness. *Journal of Logic and Computation* 15(1).

Anderson, M. L.; Josyula, D.; Perlis, D.; and Purang, K. 2004. Active logic for more effective human-computer interaction and other commonsense applications. In *Proceedings of the Workshop Empirically Successful First-Order Reasoning, International Joint Conference on Automated Reasoning*.

Anderson, M. L.; Oates, T.; Chong, W.; and Perlis, D. under review. Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance.

Coddington, A. 2007. Motivations as a meta-level component for constraining goal generation. *Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems* 16–30.

Cox, M. T., and Raja, A. 2007. Metareasoning: A manifesto. Technical Report BBN TM-2028, BBN Technologies.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving. *Artificial Intelligence Journal* 2:189–208.

Hooper, J. 2004. Darpa grand challenge 2004: Darpa's debacle in the desert. *Popular Science*.

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

Stroulia, E. 1994. *Failure-Driven Learning as Model-Based Self Redesign*. Ph.D. Dissertation, Georgia Institute of Technoloy.

Sutton, R. S., and Barto, A. G. 1995. *Reinforcement Learning: An Introduction*. MIT Press.

Tinos, R., and Terra, M. H. 2002. Fault detection and isolation for multiple robotic manipulators.

Ulam, P.; Goel, A.; Jones, J.; and Murdoch, W. 2005. Using model-based reflection to guide reinforcement learning. In *IJCAI Workshop on Reasoning, Representation and Learning in Computer Games*.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8:279–292.

Wesley, J.; Don, H.; Miller, W.; and Hajek, B. K. 1995. Fault detection and isolation: A hybrid approach.

Williams, B. C., and Nayak, P. P. 1999. A model-based approach to reactive self-configuring systems. In Minker, J., ed., *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*. College Park, Maryland: Computer Science Department, University of Maryland.

Wilson, T. D., and Schooler, J. W. 1991. Thinking too much: Introspection can reduce the quality of preferences and decisions. *Journal of Personality and Social Psychology* 60(2):181–192.