Ontologies for Reasoning about Failures in AI Systems

Matthew D. Schmill[2], Darsana Josyula[1,4], Michael L. Anderson[1,3], Shomir Wilson[1], Tim Oates[1,2], Don Perlis[1], Dean Wright[2], and Scott Fults[1]

[1] University of Maryland
College Park, MD 20742 USA
perlis@cs.umd.edu, shomir@cs.umd.edu, scott.fults@gmail.com
[2] University of Maryland Baltimore County
Baltimore, MD 21250 USA
oates@cs.umbc.edu, matt@schmill.net
[3] Franklin & Marshall College
Lancaster, PA 17604 USA
michael.anderson@fandm.edu
[4] Bowie State University
Bowie, MD 20715 USA
darsana@cs.umd.edu

Abstract. *Brittleness* is a common problem among AI systems. Autonomous systems, including those that learn, may be faced with unanticipated situations that cause decreased performance, or in the worst-case, catastrophic failures from which the system cannot recover. In this paper, we describe a construct called the *metacognitive loop* (MCL) that allows AI systems to monitor their own behavior, generate expectations about their own progress and performance, and verify that they are met. When expectations are violated, the metacognitive loop attempts to reason in a domain-general way about why expectations were not met and how to recover. The basis for reasoning is a set of ontologies that encode abstract diagnosic and prescriptive processes for coping with failures.

1 Introduction

We are interested in the *brittleness* of AI systems. A system is considered brittle when unanticipated perturbations in its domain cause significant losses in performance, or worse, complete system failure. How can brittleness be avoided without asking the designer of an AI system to enumerate and plan for all possible perturbations?

It is generally agreed that AI has produced success stories in the narrow: for large variety of well-defined AI problems, there are implemented solutions using reasonably well-understood techniques. At the very least, there are promising approaches to similar problems that give us reason to expect solutions to the problem on the way.

But in the wide sense, AI is a disappointment. Although many AI systems are at or beyond human-level competence at the individual tasks for which they

were designed, AI systems are nowhere near human-level competence across the board. Importantly, this does not seem to be a matter of building a system that has numerous subsystems, one for each of hundreds or thousands of individual tasks. Humans have, in addition to all these individual capacities, the ability to do reasonably well when faced with situations we are not trained in or even familiar with. Indeed, this ability may well be key to how we are able to become trained. We recognize when we lack an ability, we reason about how to gain it, and we recognize when we possess the necessary competence. Alternatively, we recognize when it is ineffective, too expensive, or no longer important to continue. We also (sometimes) recognize when we are in over our heads, and then we ask for help or wisely give up.

Solving the brittleness problem might be characterized as the goal of wide AI: automated human-level competence in both familiar and novel situations. In general, no system designer, whether we are talking about a learning system, a planning system, or any other 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 (e.g, by Brachman [1]). 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) that, when integrated with an existing AI system (which we will call the *host*), will allow that system to become less brittle.

We have identified four primary qualifications for such a reasoning system to provide relief from brittleness. First, the metareasoner will need access to enough information about what its host is doing (its sensors, actions, and their results), so that it can detect – and then assess – when expectations about the host's behavior are being violated in a systematic and problematic way.

Second, the metareasoner must be able to make recommendations of targeted changes to the host system to address expectation violations. Here, there must be some synergy between the metareasoner and the host; either the reasoner must have enough access to the host's internals to effect the changes itself, or the host must have pre-existing mechanisms to implement general recommendations (such as "reactivate learning" or "replan") from the metareasoner.

Third, the metareasoner must be able to use the first two capabilities to monitor how well a recommended change has addressed the problem. It must integrate a failed recommendation into its reasoning process to suggest alternatives.

And fourth, all this needs to be done in real time. This paper presents our architecture, called the *metacognitive loop* (MCL), the progress we have made to date in implementing and testing it, and outlines promising avenues for future research.

2 Related Work

There has been some progress in the literature towards integrating the different subspecialities of AI to move towards the goal of artificial intelligence in the wide sense. For instance, Goodwin [2] integrates decision theory with planning techniques to create plans with higher expected utility, using a meta-level control mechanism that mediates the allocation of computational resources between planning and the expected utility calculation. Other work by Fox [3] and Leake [4] integrates learning, planning and reasoning in introspective CBR systems that can learn about (and modify) their reasoning processes as well as their domains. Wilson [5] integrates learning and reasoning to detect and respond to situations in which the reuse of experiences in case-based reasoning goes awry due to changes in task and domain characteristics. These techniques revolve around performing computations about some aspect of a system's own processing (i.e., meta-reasoning); here, metareasoning has been applied in specialized ways to deal with the brittleness problem.

Some of the more prominent systems that might be said to employ more general forms of metareasoning include PRODIGY [6], SNePS [7], and PRS [8]. The two of these that, to the best of our knowledge, come closest to our work are SNePS and PRS, each of which has some explicit knowledge of certain types of limitations, and tools for responding when those limitations are challenged. However, unlike MCL, neither has a metareasoning component that actively and continually scans the ongoing processes for anomalies in general.

Work by Stroulia [9], Ulam [10], Ashok Goel, and others is most similar to ours in spirit. This work centers around *reflective learning*: learning based on models of a system's own reasoning process. The strength of much of this work is its concentration on determining the system component responsible for failures, a task called *credit assignment*, and developing frameworks for representing reasoning that can be analyzed and reconfigured to address problems. We feel the MCL work is a strong complement to these systems. Our goal is to build a domain general metareasoner that leverages many of the lessons that the depth of their approach provides.

2.1 The Metacognitive Loop

The metacognitive loop (MCL) is a meta-reasoner that provides AI systems with a means to detect, diagnose, and recover from unexpected perturbations. The process of dealing with a perturbation consists of three steps: (i) note an anomaly, (ii) assess it (its probable cause, severity, etc.), and (iii) guide a response into place. This seems to be a much-used capacity by humans: we regularly notice something amiss and make decisions about it (its importance, how to fix it, etc.) and then choose a course of action based on those decisions.

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 its initial response has failed, it can move down its list of hypothesized responses until it experiences success.

The basic idea of the metacognitive loop – a system that manages expectations about a host system's processes, notices when they are and are not met, and reasons about what went wrong when they are violated – has been the subject of much of our recent work. In these pilot studies, the metacognitive loop was developed alongside of and integrated with the host system. MCL had knowledge of important details of how the host system worked, and incorporated that knowledge into the reasoning process. In these prior incarnations of MCL, the boundary between the metacognitive loop and its host system was not sharply defined. There, the reasoning process that moves MCL through the note-assess-guide (NAG) cycle used a knowledge-rich, rule-based approach to diagnosis and recovery.

In more current work, we have shifted the focus from developing individual MCL-endowed systems from scratch to creating a single domain-general incarnation of MCL that can be layered on top of existing systems. This focus on generality has led us to postulate three special sets of ontologies – indications, failures, and responses – that are intended to provide a very general framework for reasoning about anomalies. The ontologies codify hierarchies of knowledge about system failures that allow MCL to map host-level expectation violations to more abstract concepts that are general across domains. We will return to these ontologies, and our most current work, in section 4.

3 Pilot Studies with MCL

In previous work, we have demonstrated the utility of the metacognitive loop as a means for improving performance in potentially brittle AI systems. In one such study we deployed MCL in a standard reinforcement learner [11]. There, learned reward functions in a simple 8x8 grid world formed the bases for expectations [5]. When reward conditions in the grid world were changed, MCL noted the violation, and would respond in a number of ways appropriate to re-learning 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.

[5] Q-learning [12], SARSA [13] and Prioritized Sweeping [14] were used.

The goal of more recent work is to establish the generality of the approach by enhancing other systems with the metacognitive loop. Two recent MCL deployments – one in a game-player, and one in a human-computer dialog agent – we present here.

3.1 A Bolo Game-Player

Bolo[6] is a 2-dimensional multiplayer game where players command simulated tanks on a virtual battle field. The goal of the game is to achieve dominance on a map by capturing refueling bases and protecting them by capturing and deploying defensive sentries called pillboxes. Tanks have offensive capabilities as well as a limited capacity for terraforming to produce defensive structures for protecting assets. Tanks in Bolo can be controlled by human players or be under the control of an external program (called a "brain" by the Bolo designers.)

The Bolo domain is surprisingly rich, supplying the brain programmer with 16 sensors pertaining to the tank, 13 terrain types on a 256x256 map, and a variable-sized array of object-property sensors that changes as objects move in and out of sensor range. Building a capable Bolo player is challenging; the best existing players are generally demolished by human players and use hand-coded routines for carrying out simple tasks like taking pillboxes and building primitive defensive structures. These brains have very limited adaptivity and are easily exploited by setting traps for which they were not programmed to avoid.

We have built a Bolo player that – while not necessarily more sophisticated – is significantly more flexible than existing brains. Its control is based on simple controllers that are organized into declarative, STRIPS-style plans [15]. Plans for simple tasks, such as capturing neutral refueling bases and pillboxes, are hand-coded and provided to the Bolo player. Algorithms for modifying plans by adding steps and changing pre- and post-conditions are also provided, making our Bolo player adaptable to changing or unforeseen conditions.

The basis for our brain is a simple plan that seeks out pillboxes and captures them by moving over them. This plan allows the brain to collect neutral pillboxes successfully. We then perturbed the basic environment of the Bolo player by making some of the pillboxes "angry" (in Bolo-speak). So-called angry pillboxes have armor, meaning the Bolo player cannot move over them to pick them up. Angry pillboxes also fire on any tank within range. The simple find-and-collect plan does *not* work here, and results in the tank being destroyed.

The Bolo metacognitive loop monitors several expectations for the Bolo player, chief among them that the player expects that the tank will not be destroyed [7]. By monitoring these expectations, MCL will identify that a perturbation has occurred as soon as it reaches its first angry pillbox and is destroyed by it.

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

[7] There is a sensor for detecting this, and as with most computer games, the player is issued a new tank when its original one is destroyed.

The MCL-enhanced Bolo player has several responses when faced with a perturbation. Its two primary responses initiate replanning: one based on means-ends analysis and operator models, and another model-free response called hypothesis driven operator refinement (HDOR). These responses allow MCL to identify what has changed after the perturbation (that the pillbox is angry), select an operator to add to the simple plan (firing on the pillbox disables it), and re-test the new plan. Our MCL-enhanced Bolo player does indeed identify angry pillboxes as anomalies, and amends its plans (and operator models) as necessary to dispatch with angry pillboxes.

3.2 MCL-enhanced human-computer dialog

Another application area for MCL is natural language human-computer interaction (HCI). Natural language is complex and ambiguous, and communication for this reason 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. Both partners elicit and provide feedback as the conversation continues, and make conversational adjustments as necessary. We contend that the ability to engage in such meta-language, and to use the results of meta-dialogic interactions to help understand otherwise problematic utterances, is the source of much of the flexibility displayed by human conversation [16], and we have demonstrated that enhancing existing HCI systems with a version of MCL allowing for such exchanges improved performance.

For instance, in one specific case tested, a user of the natural-language train-control simulation TRAINS-96 [17] says "Send the Boston train to New York" and then, after the system chooses and moves a train, says "No, send the *Boston* train to New York". Such an exchange might occur if there is more than one train at Boston station, and the system chose a train other than the one the user meant. Whereas the original TRAINS-96 dialog system would respond to this apparently contradictory sequence of commands by sending the very same train, our MCL-enhanced HCI system notes the contradiction, and, by assessing the problem, identifies a possible mistake in its choice of referent for 'the Boston train'. Thus, the enhanced system will choose 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 [18].

In more recent years, we have built a dialog system called ALFRED[8], which uses the MCL approach to accurately assess and resolve a broader class of dialog issues. The system deals with anomalies by setting and monitoring a set of time-related, feedback-related and content-related expectations.

In another scenario, if the user says "Send the Metro to Boston" and ALFRED notices that it doesn't know the word 'Metro', it will request specific help from the user, saying: "I don't know the word 'Metro'. What does 'Metro'

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

mean?" Once the user tells the system that 'Metro' is another word for 'Metroliner', it is able to correctly implement the user's request [19].

4 Ontologies for the Metacognitive Loop

Moving forward, we believe the value of MCL is attached to its generality. MCL must be able to leverage abstract, domain-independent reasoning to find solutions to problems without burdening a host system designer with the task of enumerating specific ways in which the host might fail. In this end, MCL implements three ontologies that describe different aspects of perturbations 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 allows MCL to perform abstraction and reasoning using a *spreading activation*-type algorithm. [20].

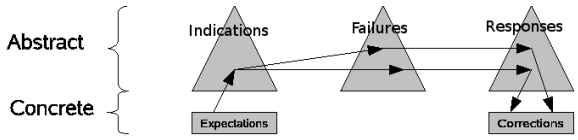

Fig. 1. Overview of the MCL ontologies for Bolo.

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 1. 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 describes means 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.

4.1 Indications

A fragment of the MCL indication ontology is pictured in figure 2. The indication ontology consists of two types of nodes: domain independent *indication nodes* shown above the dashed line, and domain-specific *expectation nodes* show 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 predictions of how sensor, state, and other values are expected to behave. Expectations are created and destroyed based on what the host system is doing. Expectations may be specified by the system designer or learned by MCL, and are linked dynamically into indication fringe nodes when they are created.

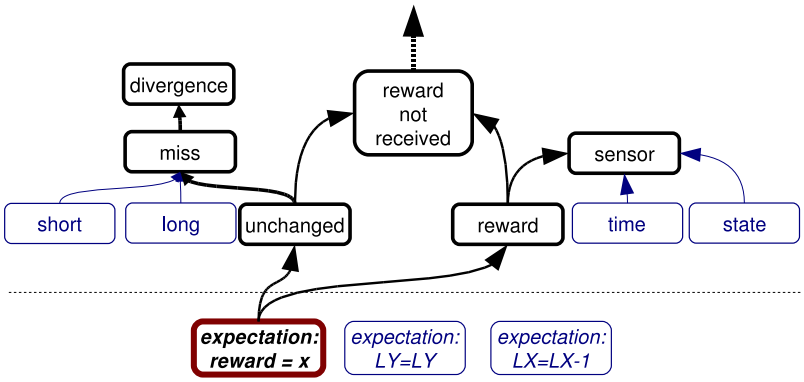

Fig. 2. A fragment of the MCL indication ontology.

Consider the ontology fragment pictured in figure 2. 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, and lastly, the sensor LX should decrease by one unit by the end of the action.

Suppose that no reward is administered, but LY and LX behave as expected. Activation in the fragment is denoted by boldface in the figure. The reward expectation node that has been directly violated is activated first. It is dynamically linked into the fringe nodes REWARD and UNCHANGED, which are in turn activated. From there, activation is propagated along *abstraction links* within the

indication core (activating the SENSOR node and others). Finally, *fringe-event links* combine the activation of individual fringe nodes into specifically indicated events. In figure 2, the REWARD NOT RECEIVED node is activated. Once all violated expectations have been noted, and activation is finished, the note phase of MCL is complete.

4.2 Failures

Once the note state has been completed, MCL moves to the assess stage, in which indications are used to hypothesize a cause of the expectation violation. 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 are 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.

Nodes in the failure ontology are initially activated 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.

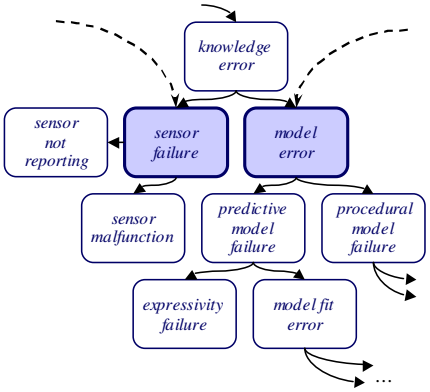

Fig. 3. A fragment of the MCL failure ontology.

Figure 3 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 initial activation in the failure ontology in our enhanced reinforcement learning example; a reward indication can be associated with either of these

types of failure. The remaining links in the figure are intraontological, and express *specification*. 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 specification links in the ontology to express this relationship.

As in the note phase, reasoning follows a kind of spreading activation, where inference is made along specification links to activate more specific nodes based on the activation of related abstract nodes. 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 no longer fits the domain.

4.3 Responses

Outgoing interontological links from active failure nodes allow MCL to move into the guide state. In the guide state, potential responses to hypothesized failures are activated, evaluated, and implemented in order of their expected utility. Interontological links connecting failures to responses are called *prescriptive links*.

Figure 4 shows a fragment of the MCL response ontology. Pictured are both MCL core responses (pictured in italics) and host-level responses (pictured in bold), which are concrete actions that can be implemented by the host system. Host system designers specify the concrete ways in which MCL can affect changes, such as by changing Q-learning parameters as seen in figure 4.

In the portion of the response ontology pictured, prescriptive links from the failure ontology are pictured as dashed arrows. These links cause initial activation in 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 concrete responses. In our example, concrete nodes correspond to either parameter tweaks in Q-learning, or resetting the Q function altogether.

4.4 Closing the Loop

Once MCL has arrived at a concrete response in the guide phase, the host system can implement the response. In our enhanced RL example, either clearing the Q values and starting over, or boosting the α parameter to increase exploration will start the agent on the path to recovery. The third possibility pictured in figure 4, to increase ϵ, may not. This is why all the activated ontology nodes are considered hypotheses; 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 fails, MCL *re-enters* and updates the ontologies in two ways. First, it inhibits the activation of the failed response node, and feeds *back*

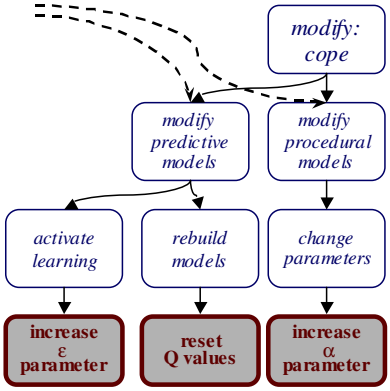

Fig. 4. A fragment of the MCL response ontology.

through the network an inhibitory signal. This dampens the activation of nodes that preceded it in the path from indications to response. Next, it feeds the new indications (those on which the failure of the response was based) into the indications ontology and the spreading activation algorithm is run to convergence. This may result in not only the original response being deactivated, but also hypotheses in the failure ontology being abandoned. The next most highly rated response is then chosen and implemented. Once a response is implemented and no new expectation violations are received, then the changes affected during the repair are made permanent, and the violation is considered addressed.

5 Bolo and Alfred Revisited

It is now possible to describe anomalies in different domains by the activation patterns they produce in the MCL ontologies. The goal in redesigning the MCL core was to identify levels of abstraction that are general enough to apply across a variety of domains and yet retain enough detail that they could be reasoned with to arrive at useful results. We will now revisit the Bolo game player and the human-computer dialog agent Alfred to illustrate how anomalies in these different systems leverage a mixture of domain-general and domain-specific knowledge to recover from perturbations.

5.1 Ontologies for Playing Bolo

To understand how the MCL ontologies apply to our Bolo player, it is important to understand the key components to the player that provide expectations and can be modified to cope with anomalies. Our Bolo player has a number of sensors available to it, including internal states and representations of the world around it. Since it is a real-time game, our player can also measure time. Expectations can be expressed in terms of those sensors, states, object properties, and time.

Our Bolo player also has a hierarchical control structure that includes primitive controllers (for moving, firing the tank's gun, and so on) and plans that sequence those controllers to achieve goals. Our player also has declarative models of its actions. Both the procedural (plans) and declarative (operator models) representations can be modified to accommodate new or changing information.

A typical perturbation our Bolo player might experience is the appearance of an "angry pillbox", which we introduced in section 3.1. Recall that a pillbox that is "angry" cannot be taken until it is disabled by firing on it, and will open fire on the player's tank when it is in range. Repeated hits by pillbox fire will destroy the tank, violating a basic expectation that our Bolo player has: it expects that its tank will not be destroyed (there is a state variable for this).

MCL signals that an anomaly has occurred as soon as the `tank-destroyed` flag is set. It enters the note phase, where the expectation violation is dynamically linked into the indication fringe based on the properties of the expectation, the sensor(s) it is built on, and the nature of the violation. In this case, fringe nodes such as STATE VARIABLE, ABERRATION (indicating that the state's value changed but was not supposed to), and BOOLEAN (indicating the state's data type) will be dynamically activated. For space reasons, these nodes are not shown in figure 2; the ontologies contain many nodes in order to cover the possible inputs an arbitrary system might have. The active fringe nodes combine in the indication core to activate a concrete indicator. In this case, that node might be called UNANTICIPATED STATE CHANGE.

The UNANTICIPATED STATE CHANGE node is linked to a number of nodes in the failure ontology. The event might be the result of a faulty state sensor (though it is not likely) or the result of a model error: either a predictive model (on which the expectation is generated), a procedural model (from which the plan is generated), or both. These candidates, and possibly others, are activated due to their relationship with the unanticipated state change node.

Finally, each of the active failure nodes will prescribe responses based on their links to the response ontology. These will be general, such as REACTIVATE LEARNING or AMEND PROCEDURAL MODELS, and not all responses will be applicable in all hosts. However, both of these nodes would be candidates in our example based on MCL's hypothesized failures. Here, MCL will attempt to refine these general responses down to concrete, actionable responses based on the specialization links within the response ontology and what it knows about the domain (and its ability to implement responses). Since our system has a planner, the nodes INVOKE MEA PLANNER[9] and REBUILD DECLARATIVE MODELS are both appropriate to the anomaly and the host system. In fact, the latter (build a model for the angry pillbox) followed by the former (build a better plan that disables the angry pillbox before attempting to capture it) is the correct course of recovery in this situation.

[9] MEA refers to *means-ends analysis*, a problem-solving algorithm. See [15].

5.2 Ontologies for Human-Computer Dialog

The job of a dialog agent is to act as an intermediary between an untrained user and an AI system (which we will call the *target*) whose native interface might be difficult for the user to learn quickly. Our dialog agent (named ALFRED, introduced in section 3.2) accepts natural language input from a user and attempts to interpret his or her instructions. ALFRED performs this task by first identifying the command (verb) in the utterance (e.g., "Send"), and then determining the objects that are involved in the command (e.g., train = "Metroliner", city = "Baltimore"). The user's command may be something for ALFRED to execute (e.g., learn a new word), or for ALFRED to interpret and execute on the target system. As ALFRED accepts instructions from the user, it attempts to maintain a declarative model of the user's *intentions*.

What the dialog agent actually does in order to act on the user's intentions depends on whether the command is meant for ALFRED or for its target system. In the former case the dialog agent is responsible for performing the action, whereas in the latter case the dialog agent is only responsible for issuing the proper command to the task-oriented system. As such, there may be expectations within the ALFRED system, the target system, and in the juncture between the two. The system can consequently be perturbed in a number of ways. Among the possible perturbations: a direct contradiction may appear in the agent's knowledge base, or the actions involved in implementing any of the sub-plans may fail, take too long, or produce unexpected results.

In the example introduced in Section 3.2 where ALFRED sends the wrong train to Boston, the dialog agent interprets the user's interjection "No" (in response to seeing the wrong train being dispatched) as the negation of his or her previous intention. This results in contradictory assertions existing in the user intention model. Since MCL expects the intention model to be consistent, this causes MCL to signal a violation. The indication fringe nodes that this anomaly will activate include STATE VARIABLE and ABERRATION (indicating that the state of the knowledge base changed from being free of direct contradictions to the state of having directly contradicting data). These active fringe nodes cause the indication core node UNANTICIPATED STATE CHANGE to become activate.

The UNANTICIPATED STATE CHANGE node will activate those nodes in the failure ontology that are related to the UNANTICIPATED STATE CHANGE node. Candidate nodes that get activated in the failure ontology include the PREDICTIVE MODEL ERROR node (referring to, in this case, the user intention model), and PROCEDURAL MODEL ERROR (in this case, referring to the commands ALFRED has executed on behalf of the user).

Each of the active failure nodes will activate the response nodes that they are linked to. Candidate nodes that are activated in this example include the corrective response nodes REBUILD PREDICTIVE MODELS and AMEND PROCEDURAL MODELS. The former will cause the dialog agent to update its user-intention model to retain the current user intention and eliminate the previous (contradictory) user intention; thereby, the knowledge base of the agent returns to an acceptable state (without direct contradictions). This constitutes a recovery

from the anomaly (in that ALFRED can proceed), but not a repair (it has not prevented the problem from occurring again).

ALFRED will then ask for a new command. If the user repeats "Send the Boston train to New York", the same processing causes another contradiction. But as MCL re-enters the ontologies a second time, the alternative failure PROCEDURAL MODEL ERROR will be the focus of attention. This will cause the response candidate AMEND PROCEDURAL MODEL and subsequently cause AMEND CONTROLLER and REVISIT INITIAL ASSUMPTIONS to get activated. It is, in fact, the assumption that any of the Boston trains would suffice that was the source of the anomaly, and revisiting those assumptions will lead to a different train to be chosen for the second "Send" instruction.

The domains of Bolo and human-computer dialog are quite different. And yet, there is a level of abstraction at which anomalies in both domains require shared concepts. The anomalies presented here, in both Bolo and ALFRED, boil down to model errors. Though the actual indications that appear with the anomalies are different, and the concrete way in which the responses are executed differ, essential parts of coping with unexpected events are domain-general.

6 Conclusions and Future Work

We have presented a metacognitive architecture for decreasing the brittleness of AI systems. We have had demonstrable success with the metacognitive loop (MCL) as an integrated component in a variety of AI domains including learning, realtime game playing, and human-computer interaction. Our ongoing work is to improve the domain generality of this tool, and in that end we have begun the development of a triad of ontologies that encode knowledge used in identifying, diagnosing, and recovering from anomalies in AI systems. Reasoning over these ontologies encompasses abstraction, specification, and inference, and allows MCL to work in a concept space that is abstract enough for a single class of indications, failures, or responses to be appropriate for application in a variety of specific domains. We presented examples of how this reasoning would proceed in two domains we work with, human-computer dialog and realtime gameplaying.

Our ontologies are still very much under development. We believe that to arrive at a high degree of domain generality, many such examples must be worked through, revising the ontologies and their linkage, and verifying that they provide adequate problem-solving behavior when the system is in operation. These ontologies and the reasoning that occurs over them are also well suited to algorithms for probabilistic reasoning on belief (Bayesian) networks. This is a topic of ongoing research, and a promising candidate for improving the ability of MCL to arrive at appropriate responses to anomalies more efficiently.

References

1. Brachman, R.J.: (AA)AI, More Than the Sum of its Parts. AI Magazine **27**(4) (2006) AAAI Presidential Address

2. Goodwin, R.: Meta-Level Control for Decision-Theoretic Planners. PhD thesis, School of Computer Science, Carnegie Mellon University (1996)
3. Fox, S.: Introspective Learning for Case-Based Reasoning. PhD thesis, Indiana University (1995)
4. Leake, D.B.: Experience, introspection, and expertise: Learning to refine the case-based reasoning process. Journal of Experimental and Theoretical Artificial Intelligence **8**(3-4) (1996) 319–339
5. Wilson, D.C.: Case-Base Maintenance: The Husbandry of Experience. PhD thesis, Indiana University (2001)
6. Veloso, M., Carbonell, J., Pe'rez, A., Borrajo, D., Fink, E., Blythe, J.: Integrated planning and learning: The prodigy architecture. Journal of Experimental and Theoretical Artifcial Intelligence **7(1)** (1995)
7. Shapiro, S.C.: Sneps: A logic for natural language understanding and commonsense reasoning. In Iwan'ska, L., Shapiro, S.C., eds.: Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language. AAAI Press/MIT Press, Menlo Park/Cambridge (2000)
8. Georgeff, M.P., Ingrand, F.F.: Decision-making in an embedded reasoning system. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence. (1989)
9. Stroulia, E.: Failure-Driven Learning as Model-Based Self Redesign. PhD thesis, Georgia Institute of Technoloy (1994)
10. Ulam, P., Goel, A., Jones, J., Murdoch, W.: Using model-based reflection to guide reinforcement learning. In: IJCAI Workshop on Reasoning, Representation and Learning in Computer Games. (2005)
11. Anderson, M.L., Oates, T., Chong, W., Perlis, D.: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance (under review)
12. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8** (1992) 279–292
13. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1995)
14. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning **13** (1993) 103–130
15. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving. Artificial Intelligence Journal **2** (1971) 189–208
16. Perlis, D., Purang, K., Andersen, C.: Conversational adequacy: mistakes are the essence. Int. J. Human-Computer Studies **48** (1998) 553–575
17. Allen, J.F., Miller, B.W., Ringger, E.K., Sikorski, T.: Robust understanding in a dialogue system. In: Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96). (1996) 62–70
18. Traum, D.R., Andersen, C.F., Chong, W., Josyula, D., Okamoto, Y., Purang, K., O'Donovan-Anderson, M., Perlis, D.: Representations of dialogue state for domain and task independent meta-dialogue. Electronic Transactions on Artificial Intelligence **3** (1999) 125–152
19. Josyula, D.P.: A Unified Theory of Acting and Agency for a Universal Interfacing Agent. PhD thesis, Department of Computer Science, University of Maryland, College Park (2005)
20. Anderson, J.: A spreading activation theory of memory. Journal of Verbal Learning and Verbal Behavior **22** (1983) 261–295