Abstract

Artificial agents need to adapt in order to perform effectively in situations outside of their normal operation specifications. Agents that do not have the capability to adapt to unanticipated situations cannot recover from unforeseen failures and hence are brittle systems. One approach to deal with the brittleness problem is to have a metacognitive component that watches the performance of a host agent and suggests corrective actions to recover from failures. This paper presents the architecture of a metacognitive agent that can be integrated with any host cognitive agent so that the resulting system can dynamically create expectations about observations from a host agent’s sensors, and make use of these expectations to notice expectation violations, assess the cause of a violation and guide a correction if required to deal with the violation. The agent makes use of the metacognitive loop (MCL) and three generic ontologies -- indications of failures, causes of failures and responses to deal with failures. This paper describes the work undertaken to enhance the current version of an MCL based agent with the ability to automatically generate expectations.

Introduction

Software systems suffer from the brittleness problems; they have limited ability to adapt to the slightest perturbations that pushes them outside their respective specifications. Artificial agents that are brittle cannot and are unable to reflect the flexibility of biological systems. To emulate the higher level thinking that gives biological systems the flexibility to adapt to changes has always been one of the goals of artificial intelligence research. The multi-level mind model proposed in (Minsky, 2006), describe the possible levels of control and management in the mind and the relationship of response time and cognitive processes. The multi-level mind defines six levels; Instinctive, Reactive, Deliberative, Reflective, Self-Reflective and Self-Conscious. Existing research has produced artificial agents with sophisticated instinctive, reactive and deliberative capabilities. To address brittleness, artificial agents need to function at least at the reflective and self-reflective levels. While performing its specified function, an agent needs to notice the perturbation, assess the situation and guide a solution into place. (Anderson & Perlis, 2005) defines this self-adapting learning capability as the metacognitive loop (MCL). MCL focuses on the reflective, self-reflective, self-conscious capabilities that enable breaking through brittleness.

Several applications of MCL exist that improve the performance of the underlying cognitive agent. In the Air Traffic Controller (ATC) (Josyula, Hughes, Vadali, & Donahue, 2009) and the Natural Language Processor (Alfred) (Josyula, Fults, Anderson, & Wilson, 2007) implementations, MCL is implemented as a component within the host agent. In the Mars Rover (Wright, 2011) implementation, MCL is an external component that controls the behavior of the host agent. Agents equipped with MCL have the ability to recover from unanticipated failures.

The interface between the host and the MCL enables a conversation between the MCL and the host agent; labeled Monitor from the agent to the MCL and Control from the MCL to the agent as illustrated in Figure 1. In the current version of MCL, MCL2, the conversation between MCL and the cognitive agent is essentially synchronous, that is, the cognitive agent invokes MCL2, waits for a response and responds with feedback on the results.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
If only synchronous communication is allowed between MCL and host, either the cognitive agent has to continually invoke MCL using Monitor call with every new observation, or, have sufficient intelligence to detect anomalies and call Monitor only when an anomaly occurs. Requiring the agent to invoke MCL introduces significant constraints. In particular, MCL misses any important factors that lead to the failures before it occurs. Therefore, it is important to allow asynchronous communication between MCL and the host.

Another major limitation of MCL2 is that expectations are specified when the host is connected to MCL. Therefore, the metacognitive component cannot learn new expectations directly by observing the host’s behavior.

To deal with these limitations, MCL3 provides two enhancements to MCL2 – (i) allow asynchronous communication between MCL and the cognitive agent by making the host’s observation stream transparent to MCL and (ii) automatically derive expectations from the observation stream for the purpose of creating an episodic memory that assists with response generation.

**Related Work**

There are several efforts ongoing to develop less brittle, perturbation tolerant artificial systems. In addition to the metacognitive loop described earlier, well known frameworks such as Soar (Laird, 2012) address the same problem.

Procedural Reasoning System (Ingrand, Rao, & Georgeff, 1992) work on a library of plans called knowledge areas (KAs) Agents based on this architecture can construct and act upon partial plans, pursue goal-directed tasks while being responsive to changes in the environment. The metacognitive ability is achieved by user defined meta-level KAs. Because of this user-reliance, the architecture is only as good as the meta-level KAs that the user specifies. Thus, there is no dynamic monitoring and correction in the true sense. If proper meta-level KAs are not specified potential problems could arise especially when multiple tasks are executed at the same time.

The Belief-Desire-Intention (BDI) agents by (Bratman, Israel, & Pollack, 1988) make use of an opportunity analyzer to propose plan options in response to perceived changes in the environment. A component checks which options are compatible with the agent's existing plans and passes on the surviving options to the deliberation process. The opportunity analyzer and the compatibility filter together provide the metacognitive ability to the underlying agents. The expectation generation module in our architecture provides functionality similar to these two components.

Recent work using Soar (Laird, 2012) enhances the performance of agents by providing them with episodic memory to support cognitive capability (Nuxoll & Laird, 2012). OpenCog Prime (Goertzel, 2009) is an architecture that builds on Cognitive Synergy Theory that defines several types of memory to support the cognitive processes.

The opportunistic control model (Hayes-Roth, 1993) describes an architecture that combines reactive (run-time event driven actions) and planning (strategic goal driven actions) approaches in a manner consistent with the intended evolution of MCL.

Further research continues on MCL, in particular, with the advent of the metacognitive integrated dual-cycle architecture (MIDCA) (Cox, Oates, & Perlis, 2011). MIDCA encompasses both the cognitive and metacognitive agent. MCL3 is well suited to operate as the meta-level process.

The integrated cognition framework (Rolfe & Haugh, 2004) organizes the essential capabilities of human-level cognition. It provides a common frame of reference for comparing cognition architectures. Along the six dimensions of the model, MCL3 is designed to provide the higher capabilities. For example, along the multi-level mind dimensions, MCL3 provides for the reflective, self-reflective and self-conscious levels. The agent that utilizes MCL3 provides the instinctive, reactive, and deliberative capabilities. Since the agent is separate from MCL3, the specification of a standard interface between MCL3 and the agent is critical.

The improved communication interface for the new MCL builds on work in agent communication languages such as KQML and FLBC. These languages focus on data passing efficiency which is relevant to the real-time exchange of observations and control messages between the MCL and the agent (Moore, 1999).

In addition, FIPA ACL and SL define standard agent communication and semantic languages, respectively, in a closely related domain (Raja, Suguri, Bloodsworth, & Khalid, 2008). The asynchronous interface design discussed in this paper is based on service oriented architecture. Other research also pursued a similar approach using CORBA (Wu, Zhou, Sun, & Yu, 2003).

**MCL2**

Agents may fail due to infinitely many specific causes. However, at higher levels of abstraction, there may only be a limited number of types of failures, causes of failures and mechanisms to recover from failures. Based on this hypothesis, (Schmill, et al., 2008) created three generic ontologies: Indications, Failures and Responses (See Figure 4). These ontologies were implemented as a Bayes net to provide MCL capability to any host that the Bayes
net is connected to. The fringe nodes in the indications and responses parts of the Bayes Net are host dependent. The fringe nodes in the indications side correspond to host-specific expectation violations. The fringe nodes in the responses side correspond to host-specific corrective actions.

In MCL2, the version of MCL used by the Mars Rover (Wright, 2011), the fringe nodes in the indications and response ontologies are hand-coded by the designer. The underlying host system issues a monitor call with the current set of observations to activate the fringe nodes in the Bayes Net. If an expectation violation is noted in the set of observations passed from the host, then the corresponding fringe node in the indications side of the Bayes net gets activated. This activation transmits through the 3 ontologies firing connected nodes. The fringe node on the response side of the ontology with the highest utility value becomes a suggestion for the host to consider.

MCL2 uses a functional Bayesian ontology for indications, failures and responses. At initialization, the user defines the agent’s primitive functions, the environmental properties that the agent can observe and report to MCL2, and, the agent’s expectations in terms of observations. At runtime, the conversation is synchronous. The agent initiates the conversation by providing current observations through the call labeled Monitor on Figure 1. MCL2 responds by providing guidance with a corrective action if there is an expectation violation through the call labeled Control. Finally, the agent reports on the success or failure of the guidance through the call labeled Feedback. MCL2 operates synchronously in that it responds to the agent’s invocations. MCL2 does not interrupt the agent and does not proactively control the agent.

An expectation is a projection of the state of observations. The agent communicates these projections at initialization. MCL2 notes deviations between the projection and the actual observations and classifies the violation using the Indications ontology. It determines the most likely cause of the violation using the Failures ontology. It uses the Responses ontology to provide guidance to the agent.

Expectations that relate to the same observations are grouped. An expectation group can be created or closed (aborted) at any time. MCL2 supports the definition of four types of expectations, based on when the expectation tests are performed:

- **Effect**: Checked when the expectation group is closed normally.
- **Maintenance**: Checked when the agent requests that the MCL compare expected observations to actual observations.
- **Delayed Maintenance**: Same as maintenance but the check is performed at a specified future point in time.
- **Temporal**: The exception is triggered because the expectation group was not closed by the specified future point in time.

MCL2 contributes a robust and flexible response determination mechanism using the three ontologies, to the problem of perturbation tolerance.

### MCL3

#### Overview

MCL3 decouples the metacognitive agent from the cognitive agent. The interaction between MCL3 and the cognitive agent is asynchronous. The integration procedure requires the specification of an Environmental interface. The interface is an XML document that defines the environment and the characteristics of the agent to MCL3. Once this initial interaction is complete, MCL3 and the agent exchange messages asynchronously. MCL3 and the agent do not engage in a synchronous conversation.

Since MCL3 maintains state and operates independently of the agent, MCL3 is able to derive expectations dynamically. These derived expectations relate to order, timing and trend of observations. Order pertains to a sequence of observations. Timing pertains to the amount of time elapsing between observations. Trend pertains to the change in the state of an observation over time.

In order to derive expectations, MCL3 groups observations into episodes and performs machine learning algorithms against sets of episodes.

MCL3 reuses MCL2’s Bayesian ontology while introducing new functionality to achieve the objectives described above. This new functionality is responsible for:
- managing the XML specification,
- interacting with the agent via messages,
- maintaining the persistent state information necessary to manage derived expectations, and,
- improving the Bayesian ontology over time.

The following diagram revisits Figure 1:

![Figure 2 – Cognitive Agent Augmented with MCL3](image)

The new metacognitive agent now consists of MCL3 that fully encapsulates the Bayesian ontology. Therefore, MCL3 adjusts its own ontology based on observations. The diagram represents the Monitor and Control calls as dashed lines to indicate they are messages asynchronously flowing between MCL3 and the agent. Feedback flows from MCL3 to the MCL2 ontologies.
MCL3 monitors four types of expectations. The user declares the first type through the environmental interface. Declared expectations describe conditions the user expects the agent to encounter. MCL3 generates the other three types of expectation; derived, refined and preventative.

Derived expectations are based on the observations of the environment and the results of control suggestions MCL3 makes to the agent. Every time MCL3 makes a control suggestion to the agent, it derives a new expectation of the outcome and uses its success or violation to adjust the ontologies.

Refined expectations create new versions of existing expectations that use more specific trigger conditions. Refined expectations allow MCL3 to train and adjust the Bayesian ontology more precisely.

Preventative expectations detect pre-conditions to the violation of the other types of expectation, as a way of preventing a violation from occurring.

MCL3 builds on MCL2 by introducing self-adjusting ontologies and asynchronous message based interface. This section decomposes the box labeled MCL3 on Figure 2. Referring to Figure 3, there are several components described in this logical design. During implementation, some components may be collapsed.

**Data Structures**

MCL3 makes uses of two types of data structures; a database and a topic. A database is a relational data store that can be implemented in memory or in an RDBMS. The design uses only one database called SpecDB. A topic is a queue that enables one-to-many distribution of messages. The author of the message is said to publish the message. Any number of recipients can subscribe to the topic and receive each new published message, guaranteed once and in publication order. If the publisher is also a subscriber, it does not receive its own publications to the topic. The design uses two topics; Observations and Suggestions. The use of the topic for the two data stores further facilitates the addition of instrumentation to MCL3.

The agent supplies observations to MCL3 on a continuous, asynchronous basis. There are two types of observations; feature and action. A feature observation reports the current state or value of a feature of the environment or of the agent itself. A feature type is defined in the Environmental Interface where it is associated with the sensor(s) that report it. An action observation reports a primitive action the agent has performed. A primitive action is also defined in the Environmental Interface where it is associated with the actuator(s) that perform it.

![Figure 3 - MCL3 Design](image)

When MCL3 detects an expectation violation, it produces a suggestion. As depicted on Figure 4, the suggestion is the outcome of using the indications ontology to classify the anomaly, the failures ontology to classify the cause and the responses ontology to identify the suggestion. The designer specifies a plan of actions the agent performs in response to each suggestion. For example, the agent may send an e-mail to an administrator in response to the Solicit Help suggestion.

**Processes**

At initiation, the agent supplies the Environmental Interface Specification (EIS) document to the Agent Specification Interpreter. The EIS defines the format and language of the Monitor message. The specification interpreter component validates the document using an XML schema. It loads the validated specification into the SpecDB. The Agent continually supplies the Monitor message into the Observations topic.
The expectation monitor component obtains goals and expectation specifications from the SpecDB. It subscribes to the Observations topic and analyzes observations to detect anomalous features from the environment that indicate expectation violations. A violation can be noted due to either the presence or absence of an observation. Upon detecting a violation or an achievement, the expectation monitor notifies the Response Evaluator. If a violation was detected, the expectation monitor suggests that the agent report actions and features as observations so that it can begin to build an episode.

The response evaluator uses the Bayesian ontology to identify the correct suggestion to provide to the agent. It immediately publishes the suggestion into the Suggestions topic. From this topic, the Agent Response Interface module packages the suggestion in a format and language specified in the EIS. It delivers the message in a format also specified in the EIS.

Simultaneously, the expectation monitor also receives the suggestion and creates a derived expectation for the suggestion. The derived expectation is based on environment features the suggestion affects. The environment features are defined by the case associated with the expectation violation.

The Case Based Reasoning (CBR) component analyzes the episodes and groups them by feature anomaly. A group of episodes related by feature is called a case database. From a case database, CBR identifies the most significant observations common to successful episodes. These significant observations define the features that the expectation monitor uses to create derived expectations.

In addition, CBR maintains parameters about the case to inform the response evaluator and the expectation monitor. CBR tracks the success and failure outcomes of suggestions by case. This information is used to build the confidence matrix for the cases. Response Evaluator uses the confidence matrix to determine when it should restructure the Bayesian ontology. CBR also measures the length of episodes over time. The expectation monitor uses this information to create deadlines in derived expectations so that it can detect a timeout failure.

Therefore, MCL3 always responds immediately with a suggestion upon detecting an expectation violation. It continually monitors observations to detect violations. It analyzes episodes to create a case. It matches a violation to a case to determine a confidence level in the suggestion. If confidence is low, MCL3 improves itself by changing the topology of the ontology when necessary.

MCL3 operates asynchronously and in parallel with the Agent. MCL3 can now initiate the Control side of the conversation without being dependent on the agent.

### Expectation Generation

MCL3 receives observations from the cognitive agent. There are two types of observations: feature and action. A feature observation reports the state or value of a feature that describes the environment or the agent. An action observation reports a discrete action the agent has performed.

An expectation is a collection of logically grouped feature-value or feature-state pairs. This collection is referred to as the expectation signature. An expectation violation occurs when an observed feature is found anomalous with respect to the features defined in the expectation. The designer specifies declared expectations in the interface. MCL2 only supports declared expectations; it does not modify or generate any new expectations. As a result, any violation conditions that arise outside of the set of declared expectation results in a default last resort response such as Solicit Help.

MCL3 uses its episodic memory to create three kinds of expectations; derived, preventative and refined. MCL3 creates a derived expectation whenever it makes a suggestion. Since the suggestion affects certain known features, including the anomalous features, MCL3 creates an expectation of which features should be affected and a deadline by which the effect should be detected in the observations.

MCL3 creates preventative expectation when it detects a certain expectation is routinely violated. Beyond a certain threshold of reoccurrence, MCL3 analyzes the episodic memory to identify observations (features and actions) that are consistent precursors to the expectation violation. Upon identifying the precursors, MCL3 creates a new expectation aimed at the detection of the precursor conditions. This type of expectation prevents the target expectation from being violated in the first place.

When MCL3 processes observations, it detects an expectation violation using the expectation signature. The matching is partial but it selects the expectation violation
that most closely matches current observations. Over time, MCL3 analyzes its episodic memory to create refined expectations from the existing expectations (declared, preventative or refined). MCL3 refines expectations by removing superfluous feature-value pairs from the signature and by adding feature-value pairs that it determines are relevant to the signature. It is important to note that MCL3 does not modify declared expectations. It adds refined expectations that allow MCL3 to more precisely target node in the failures ontology.

MCL3 builds on MCL2 by enabling a capability to generate and refine expectations over time. As a result, MCL3 is not limited by the designer’s expectation of the environment. MCL3 has the ability to learn and adapt to the fullest extent sensory perception allows.

Scenario

Consider a cognitive agent that operates a vehicle such as a planetary rover. Remote observation over several years indicated the remote planet to be arid. The designers built and trained the robot in several desert environments on Earth. The robot successfully landed and began its multiyear survey of the remote planet. A hitherto unobserved phenomenon causes the planet to enter the shadow of a gas giant creating a mini-ice age that lasts a few decades. The change in climate causes changes in the terrain that the designers had not anticipated.

The rover proceeds on its survey when it runs into an ice field. The ice causes the rover to skid out of control and off its course, until it crashes. It is still functional but several expectations were violated during this incident most related to Speed and Direction. Prior to reaching the ice field, the ambient temperature dropped steadily and rapidly. However, the designer only included expectations related to the excessively high or excessively low temperatures. Since the extremes were not reached, no expectation violation occurred.

Assume the rover is equipped with MCL2. When the incident occurs, MCL2 will provide a suggestion for each expectation violation the rover reports. Most suggestions prove ineffective because the root cause is not expressed in the declared expectations. The same incident will occur at every subsequent ice field.

Moreover, because the agent invokes MCL2, there is no opportunity for MCL2 to take proactive measures. The invocation occurs after the expectation has been violated. If MCL2 were able to determine preventive steps, it would also need a mechanism for creating a new expectation or a new suggestion and action plan for the unforeseen situation.

Assume the rover is equipped with MCL3. When the incident occurs, MCL3 processes like MCL2. After a few incidents, MCL3 attempts to build a preventative expectation. It detects that temperature drops precede each episode. As a result, it creates a preventative expectation on temperature. When the temperature reaches a certain level, the expectation is violated. MCL3 uses the ontology to identify an appropriate suggestion to deal with the temperature change. After a few more incidents, MCL3 settles on Turn Around as the most successful suggestion. Since the temperature does not drop past the level, the expectation violations that lead to the incident are avoided. Because MCL3 can create expectations and interrupt the agent, it proactively modifies the agent’s behavior even in circumstances beyond the designer’s expectations.

Assume the rover encounters conditions that trigger the new preventative expectation. MCL3 makes the Turn Around suggestion. The objective of this suggestion is to affect the temperature observation. When it makes the suggestion, MCL3 creates a new expectation derived from the originally violated expectation and the chosen suggestion. The derived expectation is that the temperature should drop to a certain level. The derived expectation has a deadline based on statistics from the episodic memory of similar incidents. Either the derived expectation is met before the deadline expires, or, the deadline expires triggering a new expectation violation. This violation indicates that the effect of a previous suggestion has not manifested itself in the environment or in the agent. MCL3 uses the ontology to arrive at a new suggestion; for example, Increase Speed or Change Direction. Again, it will take several failures for MCL3 to settle on the correct suggestion.

MCL3 uses the episodic memory to make the definition of expectations as specific as possible. It does not modify an existing expectation; it creates a new version of the expectation with more precise parameters. The new more precise expectation is called a refined expectation. Assume the rover continues its survey for some time. The episodic memory now contains several episodes of the icing incident. In each case, the agent turned around as suggested. While analyzing the episodes, MCL3 detects that the carbon level feature provides high information gain for this group of episodes. It creates two new refined preventative expectations. Both inherit the temperature expectation from the parent expectation. One refined expectation is violated when the carbon level is above a certain threshold and the other is violated when the carbon level is below the threshold. MCL3 partitions the incident space of the parent expectation when it creates derived expectation. In other words, all expectations violations of the parents also violate one of its refined expectations. The benefit of refined expectations is that they optimize the use of the ontology. It is now possible for each derived expectation to produce a completely different suggestion.
In this scenario, the original expectation did not consider carbon level. The preventative expectation worked but its initial definition was too broad. When the carbon level is low and the temperature drops, conditions still do not allow for the formation of ice. However, the suggestion is still to turn around just because of the temperature change. The refined suggestion created when MCL3 noted the significance of the carbon level feature allows it to eventually distinguish between these conditions.

The scenario illustrates how MCL3 uses episodic memory to generate new expectations and to refine existing expectations to overcome its own and the underlying host agent’s brittleness.

**Communication Interface**

In MCL3, the metacognitive agent is decoupled completely from the cognitive agent. Decoupling the metacognitive agent from the cognitive agent requires the specification of two interfaces. The first interface specification defines the environment and the cognitive agent to the metacognitive agent. The second interface specification defines the structure of operational messages exchanged between the agents in real time. The first interface is called the Environmental Interface Specification (EIS) and the second interface is called the Operational Interface Specification (OIS).

EIS defines the environment in terms of the sensors and actuators the cognitive agent possesses. The sensors are defined in terms of the observations they are capable of making. A sensor can be focused on the environment or on the host cognitive agent. The actuators are defined in terms of their capabilities to affect observations. EIS also defines the expectations the designer anticipates for the cognitive agent. An expectation is defined in terms of predicates. A predicate consists of three parts; an operator (p), a feature (f) defined in the interface, and an expected value or state (v). A predicate is denoted (o, f, v) and it evaluates true or false when a specific observation (f, v) is applied. Typically, an expectation consists of several predicates such as $\pi = \{(p_1, f_1, v_1), (p_2, f_2, v_2), \ldots, (p_n, f_n, v_n)\}$. An expectation is conjunctive; all predicates must be true in order for the expectation to be deemed violated.

OIS defines the message format that will be exchanged between the agents in real time. Since the metacognitive agent does not have direct access to the environment or the sensors, the cognitive agent reports observations to the metacognitive agent. This reporting is continuous and asynchronous. The cognitive agent does not wait for a response; it simply reports its newest observations or all of its observations as they occur. There are two types of observations. A feature observation, $(f_n, v_n)$, reports the state or value of a feature of the environment or of the host itself. An action observation, $(a_n, p_n)$, reports a primitive action the host has performed. Feature and action types and their domain are defined in EIS. The reporting therefore consists of a continuous stream of observations, $(f_n, v_n)$ or $(a_n, p_n)$, in a format and language specified in EIS. For example, the observation value pairs could be fixed format, XML, FIPA, etc. delivered by callback RPC, web service call, file exchange, etc. There is no timing implied in the observation stream.

At any point in time, the metacognitive agent can interrupt the cognitive agent with a suggestion. The cognitive agent translates the suggestion into an action plan the host can perform.

The agents use FIPA performative acts to communicate:

- **Inform**: from agent to MCL3 to report observations.
- **Refuse**: from agent to MCL3 to refuse to perform an understood request.
- **Failure**: from agent to MCL3 that request failed.
- **Request**: from agent to MCL3 to send action observations.
- **Subscribe**: from MCL3 to agent to request observations.
- **Cancel**: from MCL3 to agent to cancel subscribe or request or from agent to MCL3 to cease all learning activity.
- **Not-Understood**: from agent to MCL3 that it could not perform request.
- **Agree**: response to subscribe, request and cancel.
- **Propose**: from agent to MCL3 to propose the EIS or from MCL3 to agent to propose a new response type and action plan.
- **Accept-Proposal**: from MCL3 to agent when accepting the proposed EIS or from agent to MCL3 when accepting a new response type.
- **Reject-Proposal**: from MCL3 to agent when rejecting the proposed EIS or from agent to MCL3 when rejecting a new response type.
- **Confirm**: from agent to MCL3 that it is ready for the next message when conversation mode is enabled, or, from agent to MCL3 that it has completed the actions for the specified suggestion.

Therefore, the metacognitive agent and the cognitive agent are fully decoupled. The Environmental and Operational interface specifications allow the integration of new cognitive agents without requiring any reprogramming of the metacognitive agent. The metacognitive agent can use separate computing resources from the agent. The cognitive agent may require some programming if it was not designed to report observations or to receive suggestions.
The design of the MCL3 interface introduces the possibility of deploying MCL3 in the cloud as a service. The interface allows full separation of MCL3 from the agent so that MCL3 can make use of different processing resources. The internal machine learning processing MCL3 performs could have a negative impact on the agent if they compete for resources.

In addition, MCL3 as a service supports multiple agents. For example, a single MCL3 implementation can service several robots in the field. MCL3 does not support sharing ontologies across agents. Each agent operates within its own context. Future work considers how agent ontologies could be shared to create a hive MCL concept. For example, such a hive MCL could remain in orbit while several rovers explore a planet. Even though each rover explores a fraction of the planet, the hive allows them to share the knowledge and discoveries in a meaningful way.

Conclusion and Future Work

In this paper, we presented the high level design specification for a general purpose metacognitive agent that can be connected to any cognitive agent to improve the cognitive agent’s adaptability. The metacognitive agent continually monitors the performance of the cognitive agent to learn expectations about its environment and its behavior over time. The MCL3 specification builds on the previous iteration of the metacognitive agent – MCL2.

The communication interface between the agents uses messaging. The cognitive agent uses the interface to share its perception of the environment, its capabilities within the environment and its initial set of expectations and goals. In contrast with existing implementation, the use of messaging effectively decouples the agents and enables the reuse of whole or partial specifications between applications.

The metacognitive agent uses the initial specification and ongoing evaluation of observations to create its own expectations. The metacognitive agent notes expectations violations as indications of possible failures or opportunities and suggests responses to deal with the failure or pursue the opportunity. Since the metacognitive agent operates externally to the cognitive agent, it is not necessarily limited by the same resources.

Work proceeds to fully implement the metacognition agent – MCL3 – as described in the paper. We will compare the performance of existing applications using MCL2 against the same applications using MCL3.

Acknowledgements

This research is supported in part by the Office of Naval Research grant ONR #N00014-12-1-0430.

References


