Procedia Computer Science

Volume 41, 2014, Pages 226–232

BICA 2014. 5th Annual International Conference on Biologically Inspired Cognitive Architectures

Toward Meta-Level Control of Autonomous Agents

Dustin Dannenhauer,[1] Michael T. Cox,[2] Shubham Gupta,[3] Matt Paisner[4] and Don Perlis[4]

[1]*Lehigh University, Bethlehem, PA 18015*

[2]*Wright State Research Institute, Beavercreek, OH 45431*

[3]*Thomas Jefferson High School for Science and Technology, Alexandria, VA 22312*

[4]*University of Maryland, College Park, MD 20742*

dtd212@lehigh.edu; michael.cox@wright.edu; sglucky1@gmail.com; mpaisner@umd.edu; perlis@cs.umd.edu

Abstract

Metareasoning is an important capability for autonomous systems, particularly for those being deployed on long duration missions. An agent with increased self-observation and the ability to control itself in response to changing environments will be more capable in achieving its goals. This is essential for long-duration missions where system designers will not be able to, theoretically or practically, predict all possible problems that the agent may encounter. In this paper we describe preliminary work that integrates the metacognitive architecture MIDCA with an autonomous TREX agent, creating a more self-observable and adaptive agent.

Keywords: Computational metacognition, cognitive architecture, metareasoning, long-duration autonomy

1 Introduction

The need for robust and adaptive agents that can persist in changing environments for long duration missions requires metacognitive abilities such as self-observation and self-control, because humans cannot be required (or may not be available) to supervise such systems over periods of months. We are studying a metacognitive agent comprised of the cognitive architecture MIDCA integrated with TREX. TREX is an autonomous system control framework that has shown to be robust for a number of autonomous vehicle tasks (Py, Rajan, & McGann, 2010). Currently, most missions that the autonomous agent carries out are on a scale of tens of hours (underwater deployment was on an average of 12 hours in Rajan, Py, & Barreiro, 2013). Longer deployments, when feasible, are desirable due to several probable benefits, such as the ability to achieve different types of missions of varying lengths, better data collection with less gaps, and greater ability to respond to opportunities that arise during mission pursuit. As hardware improves to provide the physical requirements to enable

Selection and peer-review under responsibility of the Scientific Programme Committee of BICA 2014

doi:10.1016/j.procs.2014.11.107

longer deployment (i.e., battery technology), improvements in software will be required for greater autonomy. This includes the ability to adjust or replace reasoning faculties such as planning and sensing in the face of unexpected anomalies or serious failure. We present ongoing work towards an implementation of an integrated MIDCA-TREX agent.

The remainder of the paper is organized as follows. Section 2 introduces the TREX agent framework including an example of a TREX agent, and Section 3 introduces the MIDCA architecture. Section 4 details the integration of MIDCA with TREX to provide more metacognitive abilities, and then Section 5 describes preliminary experiments. Finally Section 6 presents related and future work together with concluding statements.

2 TREX: Teleo-Reactive EXecutive Framework

TREX is an agent framework designed for autonomous robots. Currently it has been deployed on underwater unmanned vehicles and used on the PR2 land robot (Py, Rajan, & McGann, 2010). This framework balances the need for robotic agents to react quickly as well as the need to pursue long-term goals. This is achieved by a non-cyclic hierarchy of internal components, referred to as reactors, where each reactor has its own sense-plan-act loop. Reactors communicate with one another using constraints on state variables. State variables are visible to all reactors but are "owned" by at most one reactor. The reactor that owns a state variable is responsible for updating its value. Reactors that wish to change a state variable (i.e., the altitude of a robot) owned by another reactor post a goal, which is a constraint on a state variable. Goals can either come from other reactors or externally (a user). To make this clearer, we walk through an example of a very simple two reactor agent.

Consider a two-reactor agent that is able to interact with a simulated light fixture. Three state variables exist. *Luminance* represents the amount of light in the room and has two possible states: **Bright** or **Dim**. *Switch* represents what would be a physical switch on the light fixture and has two possible states: **Up** or **Down**. The *light* variable represents the state of the light bulb which is either **On** or **Off**. They relate to one another such that when the *switch* is **Up**, the *light* is **On** and the *luminance* is **Bright**.

Two reactors exist in the light switch example (see Figure 1). The state variables *light* and *switch* are owned by the LightReactor, and the *luminance* variable is owned by the EuropaReactor.[1] The LightReactor acts as the direct interface and the manipulator to the light fixture and the EuropaReactor acts as the part of the agent that requests whether the room is bright or dim and plans to achieve that goal accordingly. Goals are represented as constraints on a variable. For example, one possible goal the user could request of the agent is for the *luminance* variable to be in the state **Dim** for a duration of 20 ticks somewhere in the interval between time tick 10 and tick 50. Ticks are the agent's representation of time and can be grounded into actual clock cycles. To achieve this goal, the agent must ensure that the *switch* is **Down** and the *light* is **Off**. These states are achieved if at the current time, τ, the EuropaReactor (which received the Luminance goal from the user) tasks the LightReactor with the goal to have its Light variable in the **Off** state between ticks 10 and 50. The LightReactor then creates a plan within its planning window[2] to turn the switch to the **Down** state at tick 15 while observations

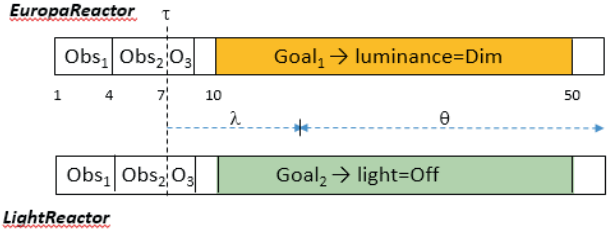


Figure 1. Two-reactor light switch example

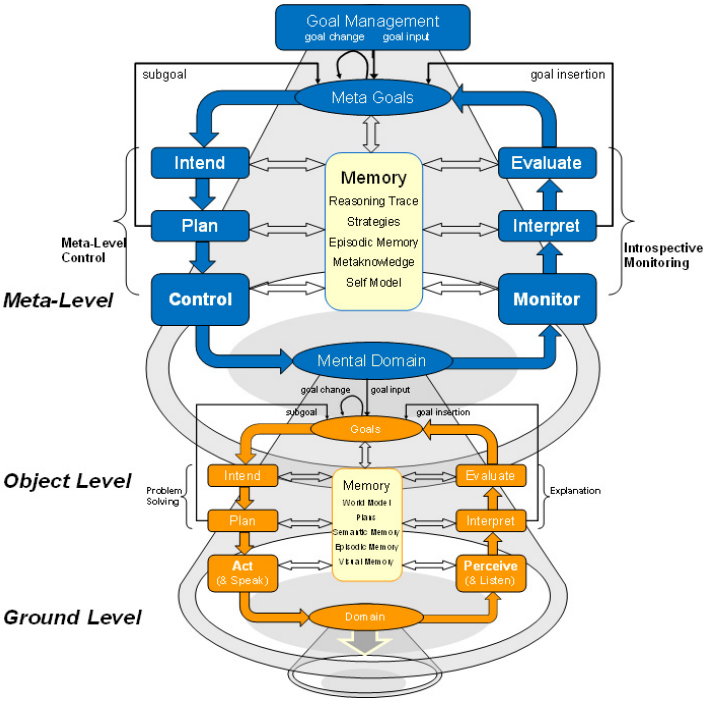
[1] The second reactor includes the Europa constraint-based planner (Frank & Jonsson, 2003)

[2] A subsequent section discusses the parameters θ and λ that together define the planning window.

are being made.

3 MIDCA: Metacognitive, Integrated, Dual-Cycle Architecture

MIDCA is a metacognitive architecture that reasons about reasoning (Cox, Maynord, Paisner, Perlis, & Oates, 2013; Cox & Oates, 2013). Metacognition has the ability to increase the performance of an agent by allowing observation and modification of the agent's reasoning components (e.g., planning and interpretation), not just its behavior. The MIDCA architecture is composed of two cycles with one at the object (cognitive) level and the other at the meta-level (see Figure 2). Each cycle consists of comprehension and problem-solving processes. The comprehension processes (on the right) begins with perception, followed by interpretation and then evaluation. At the object (lower) level, perception is of the environment in which the agent finds itself. At the metacognitive (upper) level, "perception" monitors the object level where the reasoning components are observed. This is the first part of increased self-awareness of the system because now the meta-level has the ability to detect problems in the reasoning components of the system (such as a failure in the planner or a bug in the perception of the agent). See Anderson & Oates (2007) for an extended discussion of metareasoning in artificial systems.



The problem-solving processes (on the left) begins with commitment to one or more goals (intend), planning to achieve those goal(s) (plan), and finally acting to carrying out those plans (act). At the metareasoning level, these processes are essentially the same except that actions are not to change the environment but to change (control) some part of the reasoning level.

Figure 2. The MIDCA dual-cycle architecture.

The metareasoner's job is to help improve the reasoning of the object layer. For example if a story understanding system explains an anomaly in a story but later discovers a different explanation in subsequent text, then the meta-level must take a trace of the prior reasoning that produced the faulty explanation, explain what caused the failure, and use the explanation to fix the cause of the poor explanation process (Cox & Ram, 1999). That is it repairs the interpretation component responsible for the explanation or the knowledge used to construct the explanation.

4 Using MIDCA to Parameterize TREX Reactors

All TREX reactors have two important parameters that define a planning window in the future - lookahead (θ) and latency (λ) - and which are fixed and are set by the system designer before deployment (see again Figure 1). *Lookahead* specifies how far in the future the reactor should plan, and *latency* is how much time the reactor has to finish planning. (Rajan, Py, & Barreiro, 2013). It is important that the reactors be synchronized to allow reliable behavior of the agent. Reactors that need to be most reactive (i.e., sensors or actuators) should have lower look ahead and latency (quickest to respond) while reactors that will create long-term plans should be allowed higher lookahead and latency in order to have more time to plan. Lower lookahead and latency values correlate with reactors lower on the hierarchy while higher values correlate with reactors higher on the hierarchy. For an example hierarchy, see Figure 20 (a) in Rajan, Py, and Barreiro (2013).

Lookahead and latency share a relationship with the goals of the agent. Depending on the time restraints of a goal, the lookahead and latency values need to be appropriate to allow the relevant reactors enough temporal scope and time to plan while at the same time keeping the whole system synchronized. Unnecessarily large values of lookahead means that during planning, the reactor will be looking into the future to outcomes that are not relevant and this can increase the computation overhead of the planning task. If the lookahead is too small, it will not be able to plan to achieve goals that require actions taking place at future ticks beyond the scope of the current tick + lookahead.

We suspect that as an agent is deployed over increasing lengths of time, the kinds of goals the agent pursues or the responsiveness of the agent may need to change. There may be times when the agent needs to be very accurate and the amount of time to complete tasks is less important, while at other times the agent may need to act very quickly regardless of accuracy. Under these different conditions, changing the latency and lookahead values, or the constraints of the goals, would allow the agent to respond appropriately. By integrating the metacognitive layer of MIDCA with TREX acting as the reasoning layer of MIDCA, the agent will be able to adjust reactors to better react to these kinds of changes.

In our setup reported here, we are only implementing the metacognitive layer of MIDCA (upper) and are using TREX as our reasoning layer (lower). The TREX layer is one possible implementation of the MIDCA reasoning layer. The TREX agent maintains goals it wants to achieve, it plans to achieve those goals, and concurrently perceives the world and sends these observations to the reactors of the system. Dynamically choosing good lookahead and latency values at run-time is a difficult task for a human operator, because the environment is constantly evolving even while the agent is performing its tasks. However, it is impossible to determine fixed lookahead and latency values that will be efficient across all task conditions. There are some values that may achieve the goals, but are not entirely efficient for the agent in terms of speed or response time. For example, having a small lookahead value may achieve short-term goals, but it may not be efficient for long-term goals. Our current research objective is to experiment with techniques for enabling MIDCA to determine when and to what values these parameters should change.

5 Current Evaluation and Data Collection

The model of Cox & Raja (2011) defines metareasoning in terms of introspective monitoring and meta-level control. *Introspective monitoring* provides a trace of reasoning in some knowledge representation that constitutes the observation of the object level activity. *Meta-level control* provides the ability of a metareasoner to change the future activity of the object level. Together these two sets of processes provide high level feedback to the combined agent to improve task performance. Ideally MIDCA will run together with TREX in a similar fashion in real-time to determine the effects of changing lookahead and latency values on the agent's ability to achieve goals.

In order to modify TREX in real time, MIDCA will make use of an application programming interface (API) with the TREX architecture that provides introspective monitoring and meta-level control functionality. This API exposes function calls to update latency and lookahead values, as well as read current lookahead and latency values of any reactor. The API also allows MIDCA to read and write goals to TREX which are interpreted as if they were goals from a user (the same way the agent receives goals at the start of a mission). However as this document describes late-breaking research, we currently use a manual, scripted setup to test MIDCA with TREX run individually over many instances.

The current configuration uses the light switch example described earlier to explore our ideas in a simple domain. The values for the state variables were written in eXtensible Markup Language (XML), which included lookahead and latency values for the Europa planner. The goals were also created in the same document, with fixed start, end, and duration values. These values could be changed by simply altering the XML document. To do so, a low-level XML parser was created using the existing library xml.dom.Minidom in Python. A method of the code looped over each reactor, first incrementing the lookahead value and then the latency value by 1. The start, end, and duration values were also altered. Two was added to each value, minding that the end value of the previous goal did not overlap the start value of the next goal. There was a five tick difference between the values to prevent such overlapping. Another method added goals directly in the XML document, and were monitored to prevent overlapping of the goals.

Another Python script read the log files and wrote the corresponding configuration files that contained the modified lookahead, latency, start, duration, and end values. Another Python document contained code for collecting and storing the latency and lookahead values as well as the number of goals successfully achieved. When this document ran, it started the TREX agent and created and saved a table with the values into a comma separated values file. A document was created containing 3-D bar graphs using the library MatLibPlot. Figure 3 shows a graph that was created using MatLibPlot. In this graph, the latency values range from 1 to 10. For each latency value, there are 10 lookahead values. The z-axis shows the number of goals successful, which can be 0, 1, or 2. These graphs help to determine correlations between lookahead and latency values as well as the number of goals achieved.

Looking at Figure 3 we see that for both low window-parameter values ($\theta + \lambda < 9$) and high values ($\theta + \lambda > 17$) the agent is able to achieve 2 goals, but when ($9 <= \theta + \lambda <= 17$) we see that the agent is only able to achieve a single goal. While counter-intuitive, after further analysis it was discovered that when there were low values, the planner had to be very reactive and was able to achieve planning for both goals. When there were high values the planner had enough time to explore many different possible plans fully. However, during the middle part of the graph, the values were sufficient to allow the planner to explore multiple plans (i.e., more than just reactive behavior) but not large enough to allow the planner to find the end of the correct plan in time (i.e., planning was cut off). This is the result that we observed from running the manual scripts. In our future work, this is the kind of insight (i.e., low or high values achieve more goals than middle values) we expect MIDCA to be able to learn over time in a single mission and direct the behavior of the TREX agent to either be in the low or high values, thus improving the number of goals achieved over time. MIDCA will gain both the concept and explanation of the concept, and use it accordingly.

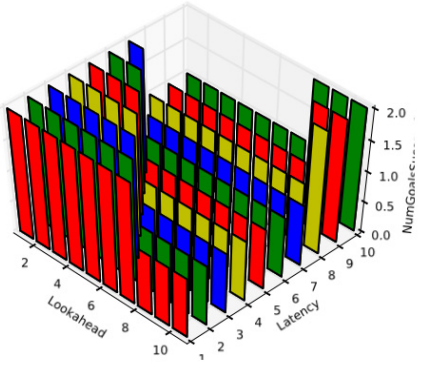


Figure 3. Number of goals achieved as a function of latency (λ) and lookahead (θ) values

6 Related Work, Conclusion and Future Work

Work on metacognitive architectures includes the CLARION architecture (Sun & Mathews, 2006) which makes use of a metacognitive component that is able to monitor and intervene in the other reasoning components of the system. While specific implementation details vary, both MIDCA and CLARION's metacognitive components are able to monitor and modify the other reasoning capabilities of the respective systems. However the objectives of the two systems are different: CLARION has a focus of replicating human-like cognition while MIDCA integrated with TREX is focused on increasing goal success within autonomous systems.

Another current metacognitive architecture is GMU-BICA (Samsonovich, 2009). GMU-BICA realizes metacognition without using an explicit metacognitive component that is separate from the cognitive processes. However, MIDCA, like CLARION, makes use of an explicit metacognitive component (in MIDCA this is the upper layer). The relationships between these three architectures are discussed in more detail by Caro, Josyula, Cox, and Jimenez (2014).

The implementation of autonomous agents performing long-duration tasks has been explored with the use of an interaction between the MIDCA architecture and the TREX agent. This paper discusses the utilization of a metacognitive component to aid a cognitive component by adjusting its reasoning modules (i.e., TREX reactors) with respect to goal achievement. An agent that is able to detect unexpected changes and adjust itself to achieve goals is an efficient autonomous agent. The utilization of such agents in the metacognitive world can significantly improve projects where agents must plan for long duration missions, like an unmanned underwater vehicle. We document our current analysis of improving a TREX agent with multiple, separate test problems. The next step in our research project is to run MIDCA with TREX automatically and in real-time as a single agent. Analysis will center on MIDCA's ability to detect the best lookahead and latency values, understand the relationship of lookahead and latency to goals, and adjust accordingly in a single execution. With more future work in the MIDCA architecture, we expect that the combined agent will be able to achieve a greater number of goals and become more responsive to change under extraordinary circumstances.

Acknowledgements

We thank Frederick Py for his assistance with the TREX software. This work was supported by ONR grants N00014-12-1-0430, N00014-12-1-0172, and N00014-13-1-0798.

References

Anderson, M. L., & Oates, T. (2007). A review of recent research in metareasoning and metalearning. *AI Magazine*, *28*(1), 12.

Caro, M. F., Josyula, D. P., Cox, M. T., & Jimenez, J. A. (2014). Design and validation of a metamodel for metacognition support in artificial intelligent systems. *Biologically Inspired Cognitive Architectures 9*, 1-23.

Cox, M. T., Maynord, M., Paisner, M., Perlis, D., & Oates, T. (2013). The integration of cognitive and metacognitive processes with data-driven and knowledge-rich structures. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*. IACAP-2013.

Cox, M. T., & Oates, T. (2013). *MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy* (Tech. Rep. No. CS-TR-5025). College Park, MD: University of Maryland, Department of Computer Science.

Cox, M. T., & Raja, A. (2011). Metareasoning: An introduction. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 3-14). Cambridge, MA: MIT Press.

Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence 112*, 1-55.

Frank, J., & Jonsson, A. (2003). Constraint-based attribute and interval planning. *Constraints 8*, 339– 364.

Py, F., Rajan, K., & McGann, C. (2010). A systematic agent framework for situated autonomous systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 2-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems.

Rajan, K., Py, F., & Barreiro, J. (2013). Towards deliberative control in marine robotics. In *Marine Robot Autonomy* (pp. 91-175). Berlin: Springer.

Samsonovich, A. (2009). The Constructor Metacognitive Architecture. In *Biologically Inspired Cognitive Architectures II: Papers from the AAAI Fall Symposium* (FS-09-01) (pp. 124–134). Menlo Park, AAAI Press.

Sun, R., & Mathews, R. (2006). Modeling Meta-Cognition in a Cognitive Architecture. *Cognitive Systems Research 7*(4), 327–338.