Towards a Generic Infrastructure to Adjust the Autonomy of Soar Agents

Scott A. Wallace
Washington State University Vancouver
14204 NE Salmon Creek Ave.
Vancouver, WA 98686
swallace@vancouver.wsu.edu

Matthew Henry
Washington State University Vancouver
14204 NE Salmon Creek Ave.
Vancouver, WA 98686
henrym@vancouver.wsu.edu

Abstract

Developing and testing intelligent agents is a complex task that is both time-consuming and costly. This creates the potential that problems in the agent's behavior will be realized only after the agent has been put to use. In this paper we explore two implementations of a generic agent self-assessment framework applied to the Soar agent architecture. Our system extends previous work and can be used to achieve adjustable levels of agent autonomy or runtime verification with only minor modifications to existing Soar agents. We present results indicating the computational overhead of both approaches compared against an agent that exhibits identical behavior without the help of the self-assessment framework.

Agents whose behavior has not been completely validated run the risk of performing their tasks incorrectly. Such situations may occur if an agent encounters situations it was not designed to deal with or if its knowledge for how to deal with a particular situation is incorrect. In the former case, the agent is operating outside of its intended (specified) scope; in the later case, the agent's implementation is inconsistent with its specification and thus incorrect. Regardless, the impact of unintended behavior may be relatively minor, or, in mission critical situations such as when controlling an unmanned aerial vehicle, the consequences may have a far reaching impact.

Prior work on adjustable autonomy (e.g., [Bradshaw & *et. al.*2005, Reed2005, Sellner *et al.*2006]) has explored a variety of approaches that allow a human supervisor, or an agent itself to deal with exceptional situations that may arise due to incomplete or incorrect knowledge. In this paper, we briefly describe a generic infrastructure of adjustable autonomy that can be used in conjunction with Soar agents and compare the performance of two implementations of this infrastructure.

Our generic framework (see Figure 1) works in conjunction with the Soar [Laird, Newell, & Rosenbloom1987] agent archicture. After a Soar agent identifies possible operators (goals or actions) to pursue next, the framework intercepts decision making to compare the actions and goals to an external policy. The result of the comparsion may deny, require or allow operators the agent has proposed. If all proposed operators have been denied, the framework provides a natural opening to defer to outside assistance. The following sections describe two implementations of this framework. They differ simply in how they interface to the Soar architecture. Below we briefy describe the implementations, comment on how existing agents must be modified to use the implementation and then examine the performance overhead they create.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

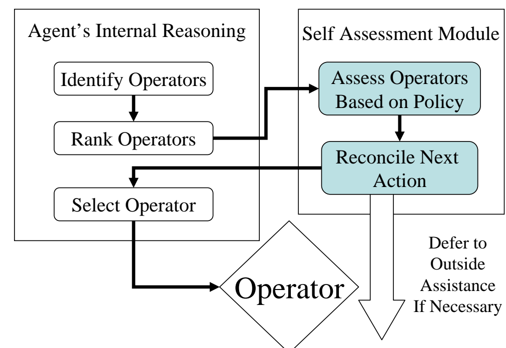

Figure 1: Generic Self-Assessment Framework

S-ASSESS

Our first implementation is implemented within the Soar agent architecture as a modular knowledge base called S-Assess which is stored in 62 domain independent Soar rules. The S-Assess library stores the agent's policy as a constraint model (CM) inside the agent's working memory. The model is loaded at run-time and can be supplied by the designer or some arbitrary third party. As the agent performs its task, S-Assess traces which goals are selected and which actions are performed. In addition, it monitors when these events take place, and the relationships between goals, subgoals and primitive actions. As it does so, S-Assess builds a hierarchical execution model of the agent's behavior. At each point in time, the agent's intended actions are temporarily added to the execution model to evaluate whether the intended action will violate the prescribed policy. If so, S-Assess prevents the agent from selecting the goal or action; if not the agent's decision process continues as normal.

Leveraging the S-Assess library requires only the avail-

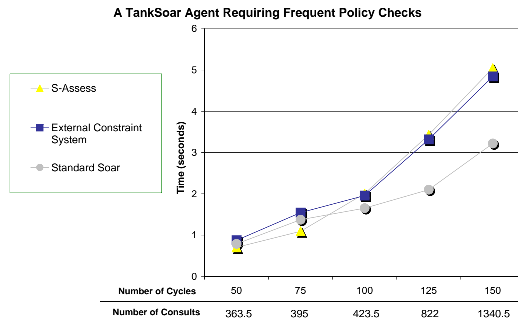

Figure 2: Runtime Performance In High-Load Situations

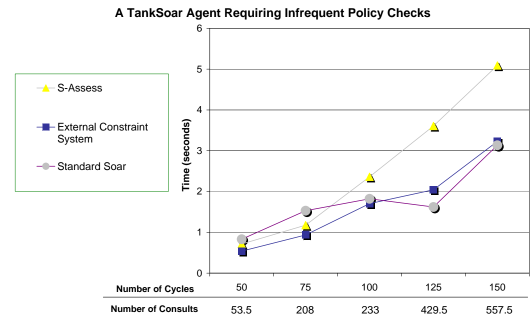

Figure 3: Runtime Performance in Low-Load Situations

ability of a constraint model and minor changes to existing agents. Specifically, we require that the agent assigns numeric preferences to operators instead of using Soar's built-in preferences. This has similar semantics, and easily allows one to establish a partial order of preferences over potential operations. However, this modification also ensures that the architecture will not commit to a specific operator as soon as the (now numeric) preferences are asserted. As a result, the assessment framework has a chance to evaluate potential options and can then assign standard Soar preferences based on the finalized rankings. Thus, by this approach, all options are guaranteed to be ranked by both the agent and the assessment framework, and the final operator selection can still be performed via Soar's standard architectural mechanism.

External Consultation System

Our second execution time monitoring system is referred to as the external consultation system (ECS). In contrast to S-Assess, this approach is external to the agent meaning that the system is written in `C` and only has access to information that can be obtained through observation or that is deliberately passes by the agent to the monitor. ECS is written in does not have direct access to arbitrary contents of the agent's working memory. The ECS loads policies directly from a text file which makes them simpler to specificy that in with S-Assess as the text representation does not have to be translated into a format that can be used within Soar itself.

Leveraging the ECS is also somewhat simpler that S-Assess. Because the framework exists outside of Soar, the agent's own rules do not need to be modified. Instead, a small additional set of *Matt: how many?* rules is added to the existing agent that forces interaction with the external framework. This interaction (called consulting) happens once for each operator the agent considers. Thus overhead associated with the system should grow as a function of how much consulting takes place.

Evaluation

The two implementations were examined within the TankSoar environment (a standard domain distributed with the Soar architecture). We tested a series of agents within the environment, stochastic elements from the environment were removed so as to make the results easily reproducible. Figures 2 and 3 illustrate the runtime results of running one agent within the testbed. The figures represent a high-load (lots of operators to check against the policy) and low-load (few operators to check against the policy) situations respectively. Each graph contains three lines: one indicating the runtime overhead each implementation while the third (labeled *Standard Soar*) indicates the performance of a Soar agent without runtime monitoring (here the Soar agent adheres to the policy without external control). The key observation from these figures is that the overhead associated with using runtime monitoring need not be prohibitive, and in relatively low-load situations can approach the performance of a Soar agent without additional constraint monitoring.

References

Bradshaw, J. M., and *et. al.* 2005. Kaa: Policy-based explorations of a richer model for adjustable autonomy. In *Proc. of the 4th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS-05)*.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *AI* 33(1):1–64.

Reed, N. 2005. A user controlled approach to adjustable autonomy. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 295b – 295b.

Sellner, B.; Heger, F.; Hiatt, L.; Simmons, R.; and Singh, S. 2006. Coordinated multiagent teams and sliding autonomy for large-scale assembly. *Proceedings of the IEEE* 94(7):1425–1444.