Investigating Ontology Infrastructures for Execution-oriented Autonomous Agents

Robert E. Wray, Sean Lisse, Jonathan Beard

Soar Technology
3600 Green Court Suite 600
Ann Arbor, MI 48105
{wray,lisse,beard}@soartech.com

Abstract

Ontologies provide useful technology for organizing and managing large-scale knowledge bases and enabling interoperability in heterogeneous agent environments. However, autonomous systems require not only large knowledge bases and knowledge sharing; they also require efficient run-time performance. In agents optimized for performance, control structures and domain knowledge are often intertwined, resulting in fast execution but knowledge bases that are brittle and scale poorly. Our hypothesis is that combining ontology representations and tools with agents optimized for performance will capitalize on the strengths of the individual approaches and reduce their weaknesses. Our strategy is to use automatic translators that convert ontological representations to agent representations, hand-coded agent knowledge for ontological inference, and explanation-based learning to cache ontological inferences. The paper outlines the rationale for this approach and design decisions and trade offs encountered. We also discuss criteria for evaluating success and understanding the consequences of design decisions on agent performance and knowledge base manageability.

Introduction

Agents must act responsively, appropriately, and robustly to the complexity inherent in their environments. Agents that can find, evaluate, and apply the right knowledge for the current situation, acting without human supervision or intervention, can be considered autonomous. Because of the primacy of responsiveness as a requirement, many agent frameworks are procedurally oriented, focused on providing agents with a robust, high-performance execution platform. Examples include BDI architectures (Georgeff & Lansky, 1987; Huber, 1999; Rao & Georgeff, 1991), Soar (Laird, Newell, & Rosenbloom, 1987; Newell, 1990; Wray & Laird, 2003), and 4D/RCS (Albus, 2001). Such agents have been demonstrated in a spectrum of high-capability, high-performance environments; however, building and maintaining such agents is resource-intensive. A drawback of procedural systems is that execution knowledge often combines control knowledge and declarative domain knowledge. While these systems execute tasks efficiently, scaling their knowledge bases to larger applications is difficult (Wray & Laird, 2003).

Ontologies (Uschold & Grüninger, 1996) can be pivotal tools for addressing the limitations of procedural systems. Ontologies are specifications of the terms used in a particular domain and their relations to other terms. Potentially, ontologies provide: knowledge scalability (the ability to encode and manage very large knowledge bases), reuse of knowledge (across agents and possibly domains), increased robustness (agents can draw on ontological relationships to reason about novel or unanticipated events in the domain), and a foundation for interoperability among heterogeneous agents. These benefits together allow applications to be developed more quickly, maintained less expensively, and adapted to new tasks more readily.

Ontology languages and tools focus on definitions and relationships more than the application of this knowledge in the execution of agent tasks. For information retrieval and web based applications, the performance costs of using wholly declarative approaches may be acceptable (Huhns & Singh, 1998). However, in performance environments, such as robotics and real-time simulation, agent responsiveness is an important requirement. At present, procedural systems fill this application niche.

There is an inherent tension between the execution performance of an agent framework, and the costs of creating and managing knowledge within that framework. This paper describes an on-going attempt to preserve the benefits of procedural agents while reducing knowledge management costs. We describe an agent infrastructure with four components:

1. Ontologies for domain knowledge representations
2. Translators that bring ontology knowledge to agent applications
3. Hand-coded ontology inference knowledge for ontological reasoning
4. A learning mechanism that caches responses to queries, obviating the need for repeated inference in response to a repeated query

Our hypothesis is that combining ontological representations and tools with agents optimized for performance will capitalize on the strengths of the individual approaches and reduce individual weaknesses. In this paper, we discuss design decisions and trade-offs when implementing this approach for agents that use the Soar agent architecture. While some design decisions and specific tools described depend on Soar, the general approach and design do not. We also describe the currently implemented solution. Finally, criteria for evaluating success and understanding the consequences of design decisions on agent performance and knowledge base manageability and scalability are introduced.

Mixing Declarative & Procedural Knowledge

We have been investigating ontology tools and languages to use in conjunction with Soar agents. Soar agents have been developed for complex, highly interactive domains such as tactical aircraft pilots and controllers for military air missions in real-time distributed simulation environments (Jones et al., 1999) and intelligent adversaries in interactive computer games (Laird, 2001; Wray, Laird, Nuxoll, & Jones, 2002) among others. The design of Soar is based on functional principles such as associative memory retrieval, blackboard memories for state information, and goal-driven processing (Newell, 1990). A Soar agent's knowledge is stored in the form of productions. Productions are rules that specify some predicate conditions and actions; actions change the state when the predicate conditions are satisfied. Production systems can be used to realize a variety of control structures (Newell, 1973). This flexibility, along with efficient pattern matching via the RETE algorithm (Forgy, 1982) and sophisticated truth maintenance (Laird & Rosenbloom, 1995; Wray & Laird, 2003), make Soar a good tool for creating high-performance agent systems.

Production conditions test both the state of an executing procedure and declarative knowledge that provides constraint and rationale for the procedure. For example, when a aircraft pilot agent (Jones et al., 1999) is maneuvering to launch a missile, productions used in the execution of this behavior test if particular altitude and heading objectives have been reached. The specific values of these goals depend on characteristics of the aircraft being flown, on the particular target, and on the weapon chosen. In naïve implementations, declarative facts such as the allowed altitude separation for launch are represented directly in the rules themselves. In more refined formulations, declarative facts are represented elsewhere in agent memory and are referenced indirectly. Additional rules encode the relevant details of each weapon and aircraft and place them into Soar's blackboard memory.

This second approach is superior to the naïve approach. It provides greater separation of declarative from procedural knowledge and is generally the rule-of-thumb used in the development of complex Soar systems. However, there are two obvious limitations. First, because the declarative knowledge is encoded by rules, changing the values or adding new ones requires a knowledge engineer that understands the syntax of Soar programs. Second, the refined approach can require more coding and is not enforced by developer tools. As a consequence, the convention is often violated and declarative parameters (e.g., the range of some missile) become hard-coded into rules. Obviously, this intermixing of procedural and declarative knowledge within individual rules leads to code that is very difficult to maintain, and, due to this difficulty, more brittle over time and agent evolution.

Because declarative knowledge is difficult to separate completely from the execution knowledge, it is difficult to reuse even the simple declarative specifications encoded with the refined approach (e.g., aircraft maximum speed). Different agents might draw on that same domain knowledge, but code-level reuse requires that the identical rule be applicable in the new system. Because inference is performed by rules custom-coded for the original system, such reuse is much more difficult to ensure.

Solution Design Considerations

We are adopting ontology tools and languages to address this limitation in current agent development methodologies. There are four obvious mechanisms by which a procedural agent can utilize an ontology. These mechanisms are listed in **Table 1**. The table is organized along two dimensions. First, the ontological information can be represented in the agent's dynamic memory (M) or in the agent's knowledge base (K). In Soar, these dimensions correspond to blackboard memory and production knowledge respectively. The second dimension regards whether the agent represents a complete ontology at one time (C), or incrementally accesses portions of an ontology as needed (I). We assume that incremental access can be accomplished as part of an agent's tasks; thus, access to the ontology should occur "on-line" with task execution. However, given the size of most domain ontologies, the complete incorporation of a domain ontology would usually need to be accessed and incorporated off-line from normal task execution.

The most straightforward solution is for the agent to access the ontology via queries and subsequent responses (IM). In this design option, the ontology database can be viewed as simply part of the agent's environment. The agent queries the database when it recognizes it needs information and then receives responses to the queries as external input. This solution has the advantage of existing protocols (such as agent communication languages and Jini) for locating remote databases and interacting with them. In contrast to CM solutions, this solution should scale to very large ontologies.

			Location of Ontology Representation	
			Dynamic Agent Memory (**M**)	Agent Knowledge Base (**K**)
Completeness of ontology representation	Incremental or partial ontology representation	(**I**)	On-demand access (query & response)	Translation + Learning
	Complete ontology representation	(**C**)	Ontology-to-agent translation	Translation + Learning

Table 1: Design options for agent access to ontology databases

There are three long-term disadvantages of the Incremental-Memory approach. First, agent knowledge is required to understand when ontology resources are needed, where to find them, and how to evaluate the trustworthiness of responses. Thus, this solution requires highly developed meta-awareness capabilities. Second, the ability of an agent to act correctly and/or responsively may be compromised by the network environment and access to needed information. As the ontology becomes a more critical component of the agent's reasoning, tighter integration of ontology and agent knowledge will be necessary. Third, in the case of simple queries without learning, queries may need to be repeated if memory no longer holds the answer to the prior query. This repetition can lead to performance bottlenecks and require agents to manage memory at a low level (e.g., caching common queries).

Incorporating the results of incremental accesses into the agent's knowledge base (IK) provides a solution to some of these issues. It solves the third problem -- queries would automatically be incorporated into an agent’s long-term memory. It mitigates the second: because the knowledge is incorporated into the knowledge base upon acquisition, repeating identical queries would not be necessary, resulting in less frequent reference to the external ontology. Creating agent knowledge to encode when to learn and where to find information would provide guidance of what and when to learn, difficult problems in agent learning. The primary drawback of incorporating ontology knowledge via learning is managing changes to the agent knowledge base. Changes necessitate manually removing learned knowledge or leading the agent through a process of "unlearning" previously cached ontology knowledge.

In contrast to the incremental approaches, it is also possible to incorporate complete ontologies within the agent's memory (CM) or knowledge base (CK). These solutions eliminate many of the meta-awareness and network reliability challenges. The agent can access ontology information without needing to access the external environment. Representing all the ontological information in memory (CM) limits this solution to ontologies of modest size, as the inference speed of many agents is a function of the size of memory. Because agent performance is often much less strongly determined by the total size of its knowledge base, this problem can be mitigated by incorporating the ontology information into the agent's long-term knowledge via learning (CK), using a process similar to the IK learning solution outlined above. Unlike the previous learning approach, because the agent is attempting to capture all the ontological information off-line from a performance context, a unique challenge in this learning environment is capturing the correct conditions under which the knowledge should be applied when in a performance context. This *recognition problem* is a critical issue when merging ontological knowledge with task execution and instance knowledge. The agent must encode not only the ontological information but also the conditions that would allow it to recognize that ontological information would be relevant to a future situation.

Long-term, our goal is to develop agent solutions that can accommodate all of these possibilities. We are initially focusing on representing complete ontologies in agent memory (CM). This focus avoids the necessity of solutions to on-line external access and to the recognition problem. Because we are using ontologies to improve knowledge management and scalability, the target ontologies are modestly sized. Representing these ontologies directly in memory does not adversely impact performance in Soar. This solution also provides a way to explore incremental transition of the ontology to long-term memory (IK) via Soar's native learning mechanism, providing partial solutions to the recognition problem.

DAML2Soar: A Complete Ontology/Agent Memory Solution

The CM solution requires three functional components: an ontology language, a translator that converts ontology knowledge to the agent language, and inference knowledge to extract relational information from ontological representations. Because optimal performance remains a primary goal, we encode ontological inference knowledge by hand. To further improve performance, Soar's learning mechanism is also used to cache ontological inferences. All of these components have been successfully prototyped as *DAML2Soar*, a system that uses DAML+OIL[1] for ontology representation and Soar as the agent architecture. This section outlines each component of DAML2Soar. The following section provides an example that demonstrates the role of each component in providing domain knowledge representation solutions for agents.

Ontology Language and Tools

As part of the semantic web (Hendler, 2001), many domain and higher-level ontologies have been developed in the DAML+OIL language (Horrocks, Patel-Schneider, & Harmelen, 2002). Given the widespread use of DAML+OIL and its representational power, we chose DAML+OIL for ontology representation. To create ontologies and to manage and combine web-based ontologies for our applications, we chose Protégé (Noy, Fergerson, & Munsen, 2000), a DAML+OIL compliant, open-source, Java-based ontology tool. Protégé is designed for data entry and knowledge acquisition, in combination with ontology representation.

One significant advantage of Protégé is its automated support for knowledge acquisition. Whenever a class is defined in the ontology, Protégé automatically creates a form-based data entry window for that class. The forms can be extended and customized, and exported for inclusion in other applications. This capability makes it straightforward to create tools that domain experts can use to enter ontological information. Using Protégé, experts do not require technical knowledge of the agents that will use the knowledge, nor do they need to know the details of the underlying ontology language.

The DAML2Soar Ontology Translator

We have implemented a translator that maps DAML+OIL ontologies into Soar production rules. DAML2Soar provides straightforward representation of ontology classes and relationships in Soar memory. Users control when and how often ontology information is relayed to Soar, simplifying version control and configuration management. No on-line access to Protégé (or to a Protégé server) is needed during execution. This solution limits interactions between an agent and the ontology knowledge base (transfer is one-way) and requires explicit compilation/translation during agent development.

DAML2Soar creates Soar productions that build a special structure in agent blackboard memory. This structure acts as a data interface layer used by the agent's execution knowledge to send queries to and read responses from the ontology. While the mechanism of DAML2Soar superficially resembles the Complete Ontology-Knowledge approach, it actually fits the CM approach in terms of function. The translated productions provide no solution to the recognition problem, and become immediately active when the agent is instantiated. We chose to translate to productions (rather than, for example, insert the ontology directly into Soar's blackboard) because this solution does not require run-time access to the agent or modification of the agent architecture.

DAML2Soar supports DAML+OIL classes, properties, super/subclass relations, namespaces, and a small set of queries (discussed below). It has been tested using ontologies gathered from the World Wide Web, including the Suggested Upper Ontology (SUO/SUMO) and Cyc's Upper Ontology (OpenCyc), as well as domain ontologies. The SUMO ontology, consisting of 628 classes and 17,896 slots, was translated into 629 Soar productions in 42 seconds on a standard PC workstation. Each production corresponds to a specific class from the ontology, with one "boot strap" production to create "world knowledge" and "ontology" divisions of the blackboard memory. Translation computation time does not scale linearly. The OpenCyc ontology (1694 classes and 152,333 slots) was processed into 1695 productions in 22 minutes. These results show that DAML2Soar can feasibly provide off-line translations of ontologies. Computational demands are not trivial, but are reasonable enough that changes to the ontology can be easily ported to the agents.

DAML2Soar does not currently support all aspects of DAML+OIL. Planned extensions to DAML2Soar include property restrictions, additional relations, and the representation of instances/individuals. DAML2Soar could also easily be adapted to other ontology representation languages, such as OWL, and to other production languages (e.g., CLIPS, JESS, or ACT-R).

Ontological Inference

Because the complete ontology is represented in agent memory, inference knowledge can be represented within the agent's execution knowledge. Rather than attempting to represent every possible inference, initially, we have developed a set of hand-coded rules that recognize the queries in **Table 2**, and then search the ontology to answer the queries.[2] Additional queries will be supported as additional DAML+OIL representational elements are incorporated within the translator.

Search over ontological knowledge is triggered via queries posted to a "query" structure on the "world knowledge" blackboard. Each query type is defined by a unique name,

[1] DARPA Agent Markup Language + Ontology Interface Language: www.daml.org.

[2] The productions that manage query generation, query syntax checking, and query posting, as well as the domain-general query search knowledge are hand-created, but then incorporated as a module of the DAML2Soar translator. This ensures every agent using the translator receives the domain-general components, eases agent maintenance costs as the ontology representation formats change, and immediately provides evidence of (small scale) reuse.

used by the inference productions to discern one type of query from another. When activated by a query, the inference productions search the ontology. Results are posted under the query structure. Unique tags indicate when a query is not satisfiable by the ontology, and when a query cannot be processed (e.g., syntax errors in query formation).

What are:	Is:
Superclasses-of-class X	X a subclass of Y
subclasses-of X	X a superclass of Y
properties-of X	
namespace-of X	

Table 2: Examples of queries supported by current agent inference knowledge

One of the advantages of this approach is that the importance of performing some particular inference can be considered in the overall context of agent reasoning. For example, if an agent was attempting to evaluate the best weapon and ordnance to choose for a particular target and it recognized that it had come under fire itself, it could deliberately choose to make activities related to evasion more important than reasoning related to weapon selection. This prioritization requires additional knowledge.

Caching Ontological Inference

Soar includes a learning mechanism, *chunking* (Newell, 1990), that can be easily applied to cache individual query responses. Each query triggers a Soar impasse, a situation that indicates the agent needs to bring additional knowledge to bear on the problem. The impasse leads to a new problem-solving context in which ontology search knowledge is activated. This search attempts to answer the query and resolve the impasse. The chunking algorithm identifies world knowledge elements that were used to answer a query and resolve the impasse. Once this information has been learned, any previously answered query can be re-answered immediately, avoiding the impasse and the consequent deliberation. This learning leads to the automatic integration of the declarative domain knowledge from the ontology into the agent's procedural knowledge.

Cached inferences may need to be removed or updated when the ontology changes. Currently, we delete all cached inferences when the ontology changes. However, a significantly better solution would be to identify what cached knowledge needs to be removed or updated, and what can be preserved without change. Ontology versioning solutions, along with tools that examine cached productions, could automate an analysis of which rules to retain and which to excise following ontology modification.

Current Application: Networked Command, Control and Communication

The approach outlined above is being explored and implemented for Cooperative Interface Agents for Networked Command, Control, and Communications (CIANC^3) (Wood, Zaientz, Beard, Frederiksen, & Huber, 2003), a Department of the Army Small Business Innovation Research project sponsored by the U. S. Army Research at Fort Knox. The "CIANC^3 ontology" is a collection of taxonomies, communication protocols, and deontic relationships for tactical mechanized infantry operations (Kumar, Huber, Cohen, & McGee, 2002). For example, the ontology includes descriptions of the types of vehicles one would expect to find on a future infantry battlefield, their weapons, and operational parameters (speeds, size of crew, etc). The ontology is being represented in Protégé and translated into Soar via DAML2Soar.

Figure 1 illustrates how the agent uses knowledge from the CIANC^3 ontology to perform its tasks. Production rules from DAML2Soar instantiate the ontology in the agent's blackboard memory. The ontological knowledge can be queried by searching via "standard" ontological relationships (e.g., subclass). This knowledge would allow an agent to recognize, for example, that "M1A1" is a kind of tank and that the characteristics of its primary weapon determines the maximum range at which it can directly engage hostile forces. These productions are not application or agent specific and can be used in any application using the solution presented here.

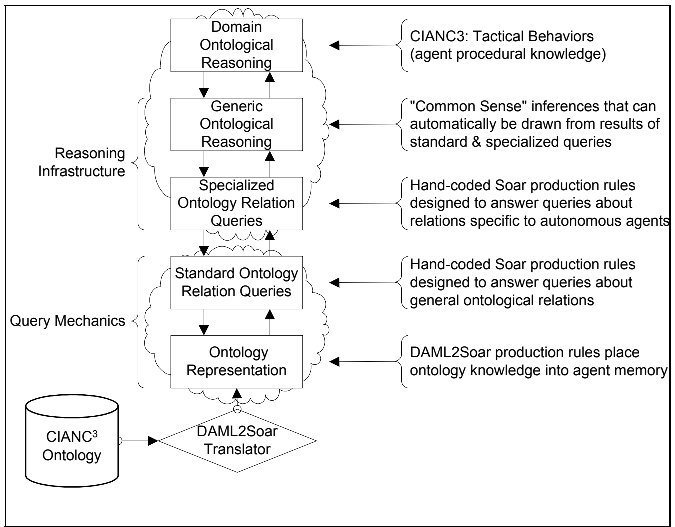
Domain Ontological Reasoning
CIANC3: Tactical Behaviors (agent procedural knowledge)
Generic Ontological Reasoning
"Common Sense" inferences that can automatically be drawn from results of standard & specialized queries
Reasoning Infrastructure
Specialized Ontology Relation Queries
Hand-coded Soar production rules designed to answer queries about relations specific to autonomous agents
Standard Ontology Relation Queries
Hand-coded Soar production rules designed to answer queries about general ontological relations
Query Mechanics
Ontology Representation
DAML2Soar production rules place ontology knowledge into agent memory
CIANC3 Ontology
DAML2Soar Translator

Figure 1. Use of the complete ontology/agent memory solution in the Cooperative Interface Agents for Networked Command, Control, and Communications application

At the next higher level, the ontology reasoning infrastructure includes productions that can reason across domain- or agent-specific relations. These production rules comprise some "common sense" reasoning for the domain and compare the results of ontological queries with the agent's mental representation of the world-state (Beard, Nielsen, & Kiessel, 2002). These comparisons allow the agent to draw further domain specific inferences on the basis of ontological relationships amongst objects represented in the agent's perceived world-state. For example, by recognizing that an M1A1's primary weapon is a direct fire weapon, the agent could determine that the tank must have a direct line-of-fire to a target for engagement of that weapon. The productions in the ontological reasoning layer have limited reusability (because the semantics of relations are defined operationally in the productions, rather than formally in the ontology), but provide a very convenient tool for expressing relationships that are difficult to express formally (such as the tactical consequences of the differences between guns and howitzers). Further, these productions can capture complex relationships that could be derived via ontological inference, but only with significant inference effort. This level thus offsets some of the performance costs to be expected when implementing queries without also using the optimizations inherent in databases. At the highest level, agents are able to evaluate their own perceived state in the context of the ontology-based retrievals and make decisions that are consistent with that world state, querying the ontology and acting based on its interpretation of the results.

Considerations for Evaluation

We have argued that the approach outlined in this paper will lead to lower knowledge management costs and increased scalability, without degrading the performance of an agent optimized for execution. We have implemented a prototype solution and are now beginning to evaluate it. Our evaluation will concentrate on measuring performance costs and knowledge representation improvements.

The first priority for the new solution is to maintain acceptable performance. The cost of agent development precludes developing both the standard agent representations and the new ontology-based representations in order to make empirical comparisons. However, we can compare the performance of existing Soar performance systems (such as TacAir-Soar) to the CIANC^3 DAML2Soar solution. Clearly, there will be some cost in the DAML2Soar system, because the agent now retrieves declarative information from the ontology rather than accessing it directly. Empirical testing will provide evidence of the extents and consequences of this cost.

Ontology agents may or may not have larger knowledge bases than typical agents, but they certainly will have larger blackboards: relationships implicitly represented in rules will now be explicitly represented in the ontology. We must determine the impact of the size of these larger blackboard memories on the performance of ontology-based agents. There will likely be a "break point" in terms of ontology size. After the break point is surpassed, performance may severely degrade. To date, we have not observed such a point with the ontologies we have tested in Soar, but, because one goal of the ontology-based solution is scalability, we must find the break point in order to understand the limits of the DAML2Soar solution.

Measuring development cost and reuse are also difficult. One particular challenge is to compare the cost of developing and maintaining ontologies to the original agent methodologies. However, qualitative measures should provide some indication of any benefit the ontology solution provides. First, any general knowledge will transfer immediately to new applications needing that knowledge. For example, the CIANC^3 ontology will include representations of authority, communications, and teamwork that should transfer to other applications needing these capabilities. For most Soar systems, even those designed as reusable rule sets, reuse without significant re-implementation has not been common. However, we have demonstrated rule-level reuse in an ontology-based communications component (Wray, Beisaw et al., 2002). If DAML2Soar improves such reuse, it will represent a categorical improvement, at least for Soar systems. Second, we can measure the ratio of reused and automatically created rules to total rules. As described previously, in most Soar agent systems, this ratio is very small, due to the intermixing of declarative and procedural representations. With the CIANC^3 agents, we will establish a baseline ratio. Over time, if ontological reuse is successful and our solution scales, the ratio should grow larger for new applications.

Conclusion

Ontologies provide the potential to improve knowledge manageability, scalability and reuse for intelligent systems. Autonomous agents that employ large knowledge bases will benefit from such technologies. In this paper, we have introduced a number of design dimensions for consideration when combining ontological approaches with procedural agent systems. One of these options has been prototyped and we are currently evaluating its potential to provide more cost-effective knowledge management while maintaining excellent performance. Preliminary results suggest that ontologies have the potential to improve reuse of agent knowledge significantly, at least in rule-based agent systems.

Acknowledgements

This report was prepared under a Department of the Army Small Business Innovation Research Program 2000.2 contract for topic A02-024 (contract #DASW01-03-c-0019). We gratefully acknowledge the sponsorship of this research by the United States Army Research Institute and thank ARI Program Manager Carl Lickteig for his guidance and support.

References

Albus, J. S. (2001). *Engineering of Mind: An Introduction to the Science of Intelligent Systems*: John Wiley and Sons.

Beard, J., Nielsen, P., & Kiessel, J. (2002, December). *Self-Aware Synthetic Forces: Improved Robustness Through Qualitative Reasoning.* Paper presented at the Proceedings of 2002 Interservice/Industry Training Simulation and Education Conference, Orlando, FL.

Forgy, C. L. (1982). RETE: A fast algorithm for many pattern/many object pattern matching problem. *Artificial Intelligence, 19*, 17-37.

Georgeff, M., & Lansky, A. L. (1987, August). *Reactive reasoning and planning.* Paper presented at the 6th National Conference on Artificial Intelligence, Seattle, Washington.

Hendler, J. (2001). Agents on the Web. *IEEE Intelligent Systems, 16*(2), 30-37.

Horrocks, I., Patel-Schneider, P. F., & Harmelen, F. v. (2002). *Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web.* Paper presented at the 18th National Conference on Artificial Intelligence.

Huber, M. (1999, May). *JAM: A BDI-theoretic Mobile Agent Architecture.* Paper presented at the Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, Washington.

Huhns, M. N., & Singh, M. P. (1998). Agents and multiagent systems: Themes, approaches, and challenges. In M. P. Singh (Ed.), *Readings in Agents* (pp. 1-23): Morgan Kaufman.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. G., & Koss, F. V. (1999). Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine, 20*(1), 27-42.

Kumar, S., Huber, M., Cohen, P., & McGee, D. (2002). Toward a Formalism for Conversational Protocols Using Joint Intention Theory. *Journal of Computational Intelligence, 18*(2), 174-229.

Laird, J. E. (2001). *It Knows What You're Going To Do: Adding Anticipation to a Quakebot.* Paper presented at the Fifth International Conference on Autonomous Agents (Agents 2001), Montreal, Canada.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(3), 1-64.

Laird, J. E., & Rosenbloom, P. S. (1995). The evolution of the Soar cognitive architecture. In T. Mitchell (Ed.), *Mind Matters*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Newell, A. (1973). Production Systems: Models of Control Structures. In W. Chase (Ed.), *Visual Information Processing*. New York: Academic Press.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, Massachusetts: Harvard University Press.

Noy, N. F., Fergerson, R. W., & Munsen, M. A. (2000). *The knowledge model of Protege-2000: combining interoperability and flexibility.* Paper presented at the Proceedings of EKAW 2000.

Rao, A. S., & Georgeff, M. P. (1991). *Modeling rational agents within a BDI-architecture.* Paper presented at the Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning.

Uschold, M., & Grüninger, M. (1996). Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review, 11*(2), 93-155.

Wood, S., Zaientz, J., Beard, J., Frederiksen, R., & Huber, M. (2003). *CIANC3: An Agent-Based Intelligent Interface for the Future Combat System.* Paper presented at the 2003 Conference on Behavior Representation in Modeling and Simulation (BRIMS), Scottsdale, Arizona.

Wray, R. E., Beisaw, J. C., Jones, R. M., Koss, F. V., Nielsen, P. E., & Taylor, G. E. (2002, May). *General, maintainable, extensible communications for computer generated forces.* Paper presented at the Eleventh Conference on Computer Generated Forces and Behavioral Representation, Orlando, Florida.

Wray, R. E., & Laird, J. E. (2003). An architectural approach to consistency in hierarchical execution. *Journal of Artificial Intelligence Research, 19*, 355-398.

Wray, R. E., Laird, J. E., Nuxoll, A., & Jones, R. M. (2002). *Intelligent Opponents for Virtual Reality Training.* Paper presented at the Inter-service/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL.