Beyond the Ad-hoc and the Impractically Formal: Lessons from the Implementation of Formalisms of Intention

Sean A. Lisse[1], Robert E. Wray[1], Marcus J. Huber[2]

[1]*{lisse, wray} @soartech.com*
Soar Technology
3600 Green Court
Ann Arbor, MI 48103

[2]*marcush@marcush.net*
Intelligent Reasoning Systems
Oceanside, CA

Abstract

This paper describes a BDI-inspired abstraction layer that is implemented in Soar to increase the scalability and flexibility of knowledge engineering in Soar-based agent systems. We motivate the BDI abstraction with a specific example of a multi-agent system that is being developed to explore human command and control of heterogeneous robotic entities. We then describe how we have implemented deontics and communication aspects of Joint Intentions via the abstraction layer and illustrate the strengths, limitations, and lessons learned in this approach.

Introduction

There is an inherent tension in the design and implementation of complex, real-world software systems and the "pure" theoretical precursors that shape and inform the implementation. In this paper, we describe an explicit attempt to move from a historically successful but informal design and development process to one motivated and informed by belief-desire-intention (BDI) theories.

A unique element of this approach is a formally-motivated BDI abstraction layer built on top of Soar, [1] an agent architecture which lacks the formal semantics of BDI architectures such as SPARK [2]. Soar has successfully demonstrated soft real-time intelligent behavior and reactive planning, and it has been effectively used in applications requiring these characteristics [3, 4].

Many Soar applications are models of human behavior. However, we increasingly apply Soar as a general agent architecture [5]: Soar's performance scales exceptionally well as knowledge stores increase [6]; its combination of operator-based reasoning and built-in reason maintenance provides powerful knowledge-engineering short-cuts for developing agents that are embedded in dynamic domains; and its associative control mechanism gives executing agents tremendous flexibility in deciding how they will address situations they encounter in the environment.

While using Soar as a general agent architecture to build large scale, general-purpose agents (i.e., those not intended as human behavior models), we have encountered a number of issues that motivate BDI-based abstractions on top of Soar:

- Soar includes some computational restrictions that are, by design, similar to those of humans [5]. Agent systems that are not models of human behavior need not impose similar limits. For example, the approach to the maintenance of multiple, simultaneous task goals (described in detail below) is motivated in large part by Soar's "cognitive bottleneck" constraint, which is derived from cognitive psychology [7]. BDI offers an approach that is less constrained by human psychology, though new approaches to multiple simultaneous goals are a subject of some on-going BDI research (e.g., [8]).
- Large-scale knowledge systems have been created with Soar, but these systems are difficult to verify and validate. Although we could build V&V tools specific to Soar agents, formal properties of BDI systems offer more direct routes to semantic verification.
- While Soar systems scale well in terms of performance with increasing knowledge, the knowledge-engineering costs of scaling to larger knowledge bases can be prohibitive. Some of the difficulty in implementing ever larger knowledge bases is due to the ad hoc nature of prior solutions developed within Soar (Jones and Wray, 2006). BDI offers a more principled approach to engineering large knowledge bases.
- BDI is the dominant agent paradigm, and we often translate BDI advances into Soar terms. Teamwork is an obvious example of this translation [9] [10]. A BDI-motivated abstraction layer should facilitate the application within Soar systems of general research advances in BDI.
- BDI includes concepts readily familiar to most people, easing comprehension of the agents by non-technical users. We have discovered in previous work [11] [12] , that the BDI abstraction is more readily understood by end-users than are the low-level details of Soar.

The hypothesis motivating this research is that a BDI-based abstraction for Soar will provide many of the benefits of BDI systems, while capitalizing on Soar's strengths in building large-scale, complex agent systems. We do not present certain confirming or disconfirming evidence of the validity of this hypothesis in this paper. Rather, we focus on the practical issues encountered in trying to apply formal BDI principals to Soar agent systems and our progress toward evaluating the hypothesis.

We first describe a specific example of an agent system. Subsequent sections then describe the developing BDI abstraction layer for Soar, using the example application to

motivate and illustrate the approach. We highlight where the BDI abstraction has provided value, where the value is less clear, and what lessons we believe are applicable to others trying to build large-scale systems based on BDI-motivated theoretical principals.

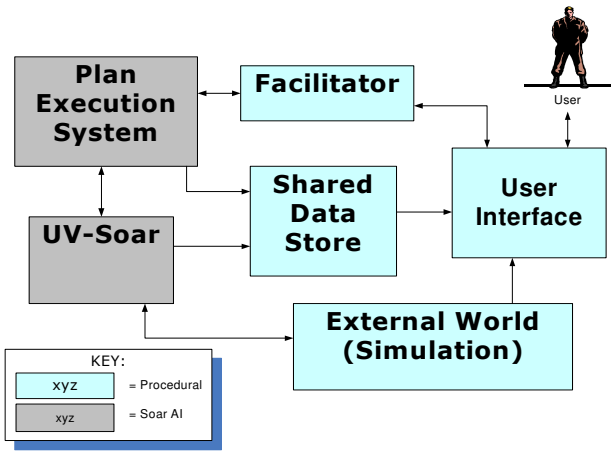

Figure 1: ICF architectural overview

Application Context: Intelligent Command & Control of Robotic Systems

Soar Technology is exploring human command and control of autonomous robots, focusing especially on cognitive overload. The "Intelligent Control Framework" (ICF), illustrated in Figure 1, applies intelligent software agent technologies to automate many tasking, monitoring, and communication activities, simplifying demands on the human operator of unmanned systems.

ICF contains a command and control user interface, a dialogue management component ("Facilitator"), a Plan Execution component, and a shared data store. The testbed includes a simulation of a battlefield with robotic unmanned vehicles (UV-Soar). The UV-Soar agents and the Plan Execution agent are built in Soar, and the rest are built in procedural languages. UV-Soar agents are conceptually similar to previous simulated, autonomous tactical vehicles built in Soar [3] but are implemented with the BDI abstractions detailed below.

The Plan Execution System acts as a central controller for the overall system, providing commands to UV-Soar agents, accepting guidance from the human operator, and sending status communications as needed to all system components. Like UV-Soar, the Plan Execution System is built on top of the BDI abstraction described below.

A key requirement for this system is to enable ready and efficient human understanding of an unfolding tactical situation. The BDI abstraction layer supports communication of the agents' intentional stance to users, by reifying agent desires and intentions.

A BDI Abstraction Layer for Soar

A BDI system should include, as part of its implementation, a declarative and introspectable representation for the system's goals [13]. "Active goals," which correspond to an intention in BDI [14], typically are represented declaratively in Soar. However, there are limitations in current approaches to the representation of task goals in Soar.

First, there are a number of different idioms [15] for representing active goals. In the most prevalently used idiom, Soar's operator/problem-space stack is used to represent task goals, leading naturally to a hierarchical decomposition of tasks similar to an HTN execution system [16] [17]. This approach employs Soar's operator maintenance functions for task goals, which can simplify knowledge engineering. Other idioms generally represent task goals as elements in memory, which then require additional explicit knowledge to activate, monitor, and deactivate the goals.

Second, all of the Soar idioms are informal—that is, there is no formal computational model of the idiom. This absence of a model leads to solutions that are not uniformly implemented (not just across agents but also within agents). The resulting variability leads to systems that are more difficult to engineer, verify, and maintain [14].

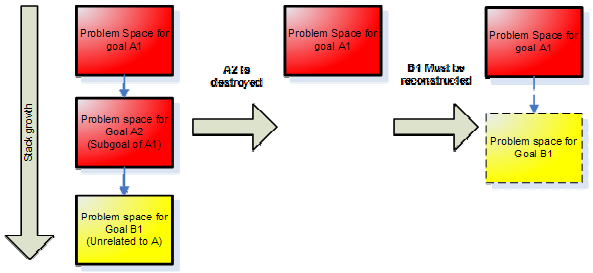

Figure 2: Destruction of unrelated goals

A specific example of the informality of the idioms is an approach called "floating goals" [18], which is used in the HTN-style idiom. Soar's subgoaling mechanism only allows a single subgoal at a time. When representing task goals on this stack, there are times that an active subgoal is unrelated to its "parent" goal but has been placed there only because that subgoal needs urgently to be addressed. As shown in the left column of Figure 2, these goals are independent of the parent goal in terms of the task but "float" below it within the goal stack[1]. A significant problem occurs when the original goal has resolved, but the unrelated subgoal has not. Soar responds to an achieved goal by removing the goal stack including the unrelated "floating goal". Since this destroys the floating goal, it must be reconstructed. If the goal context included

[1] We describe the goal stack as "growing downward," so that a lower goal is a subgoal of the higher one.

information not available elsewhere (such as a record of a past event), such reconstruction may be impossible[1]. This means that there is no way to consistently manage multiple active goals within this specific idiom in Soar.

Other idioms do allow multiple, simultaneous task goals, but these approaches are limited by a lack of detailed computational descriptions. Further, all the idioms generally assume that the conditions for activating or deactivating a goal are impenetrable, which may be a good assumption for human-like models but makes consistent knowledge engineering difficult and inhibits some flexibility (essentially, agents have limited ability to debug their own errors).

Through exploring a computational specification for an alternative idiom, we came to realize BDI provided many of the concepts needed. For example, many BDI systems assume or allow for multiple top-level goals that may be suspended and resumed without appreciable loss of context [19, 20].

In order to overcome some of the above difficulties, we have implemented a BDI-inspired Soar idiom in Soar productions. The BDI abstraction layer includes:

- Declarative *goals*
- Declarative *transforms* (plan operators)
- Goal and transform construction templates
- Goal and transform macros for recording contextual information specific to each goal or transform
- Activation pools for the goals and transforms: *desired*, *active*, and *terminated*
- (Optional) Support for declarative activation conditions controlling commitment to the goals and transforms
- (Optional) Automatic creation of Soar operators from the declarative transforms

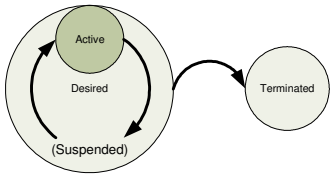

Figure 3: Activation

The implementation is supported by a set of preprocessing macros. The macros allow users to code at the BDI level, while a simple compiler expands the BDI constructs to Soar productions. The macros ensure consistency and are application-independent. Because the macros enable programming at the BDI level, we refer to this implementation as an "abstraction layer" in contrast to the prior Soar idioms, where the programmer was always working at the level of Soar productions.

Goals, transforms, and their respective activation pools and conditions all exist within Soar's memory. The main differentiators between goals and transforms are: 1) goals describe a desired world state, whereas transforms describe a particular action for bringing about that world state, and 2) transforms must be anchored to some active goal – agents are required to ground their actions in goals.

[1] As a result, floating goals are limited by convention to one Soar decision.

Figure 3 depicts the activation pool transitions for goals. The activation pools and transitions for transforms are structured in the same way, but they inhabit separate pools from those for goals.

Goals or transforms in the desired pools may be viewed as BDI desires, and goals or transforms in the active pools as BDI intentions. Goals or transforms in the active pool must also be present in the desired pool and are removed from the active pool if they are no longer present in the desired pool. This allows the library to take advantage of Soar's reason maintenance mechanisms to ensure that when the motivations behind a goal or transform no longer apply, the commitment to the goal or transform is reconsidered[2]. Goals or transforms in the terminated pool provide the agent with a historical record.

Assessment of BDI Abstraction Layer

The declarative goal layer fulfills the need for explicit declarative goals, supports goal introspection, and provides flexible goal-commitment strategies. Inference mechanisms inherent in Soar allow agents to maintain the activation states of their goals while allowing adequate performance in soft, real-time applications.

While performance remains adequate in tested applications, the BDI abstraction potentially introduces a substantial performance cost. Soar's execution speed derives largely from a specialized implementation of the Rete algorithm [6]. With the BDI abstraction, goal commitments have been moved from production rules to working memory. This declarative representation results in a (relatively) larger active memory, which slows Rete.

A more significant performance concern is the number of partial matches generated by a large set of similar memory elements, such as the goal elements created in Soar memory when using the BDI abstraction. Rete can perform very poorly in these conditions [21]. To date, partial-match cost has been lessened by manual solutions where a developer actively scans for expensive matching and develops point solutions for specific, expensive rules in the abstraction layer. Long-term, we will investigate alternatives to Rete (such as collection match) and evaluate whether Rete is an appropriate matcher given the different assumptions placed on Soar memory under this approach.

Finally, the BDI abstraction layer clearly does not represent human cognition, and no system using it can claim any validity as a model. It is an open question whether Soar, when considered as a theory of human cognition, should *allow* this BDI abstraction at all.

[2] Reconsideration in Soar serves the same functional purpose as reconsideration in BDI (Schut and Wooldridge, 2001). However, Soar uses reason maintenance rather than decision theory to effect reconsideration (Wray and Laird, 2003).

Joint Intention Theory

The BDI abstraction layer has enabled the direct realization of BDI concepts in Soar agent systems. For example, ICF requires communication and collaboration among the different role players in the system, including the human operator. Given these two goals, we looked to Joint Intention Theory (JIT) to guide the creation of agent teams, drawing especially on fundamental ideas from Cohen and Levesque [22], Tambe [10], and Huber, Kumar et al. [23]. In short, the JIT formalism specifies the conditions under which "teams" of agents are formed and dissolved. With the JIT basis, agents coordinate not simply as individuals pursuing similar goals that they may unilaterally abandon at any time, but as a team with shared understanding that is robust in the face of certain possible failures and misunderstandings. The following sections describe how we have used JIT to inform implementations of deontics and communication via the BDI abstraction layer.

Deontics

Deontics pertains to representing and reasoning about inter-agent commitments and restrictions in a multi-agent system. We have built a reasoning system that supports deontic operators for *obligation*, *permission*, and *prohibition*, as well as a hybrid operator that we call *permission-with-permission*. Permission-with-permission is a special case requiring the user's explicit permission to execute each instance of a goal or transform [24].

Each instance of a deontic operator includes a type specification (as above), a characterization of the goals or transforms that it should affect, and a relativizing condition. The relativizing condition determines the situations in which the deontic operators apply.

These deontic operators follow the activation cycle shown in Figure 3. When a deontic operator is active, it is compared to any goals or transforms within their respective desired pools. If the goals or transforms match the deontic operator's characterization filter, it is said to apply to the goals or transforms. Applicable deontic operators take effect by altering the activation conditions of the affected goal or transform.

Activation conditions are evaluated by Soar's matching system and reason maintenance upon change, and they control the activation status of the goal or transform. For example, a prohibition deontic operator will add an invariant to the goal that always evaluates to false. If the goal is in the active pool, this immediately forces the goal to transition out of the active pool (while allowing it to remain in the desired pool). The new invariant depends upon the deontic operator for support and will vanish if the operator becomes inactive.

We have also implemented JIT-based MASTER and SLAVE authority roles [24] for the agents; acceptance of orders (*requests*) depends upon the order-giver's role (hence authority) relative to the recipient of the order. The resulting capabilities model hierarchical organizations such as the military: ICF agents accept orders only from other agents with an appropriate role.

ICF is designed to support a single user controlling a set of robotic entities. The command set is currently limited to hierarchically controlled subteams. MASTER-SLAVE has proven to be sufficient for the current implementation only because the single-user/controller assumption demands only the MASTER-SLAVE relationship. Other available authority role definitions (SUPERIOR, SUBORDINATE, PEER) [24] will likely prove useful in the future as we increase the complexity and autonomy of the controlled vehicles.

To date, it is rare for an entity to fail to properly carry out its role. This robustness is due primarily to the stability and predictability of the test environment. Because of the reliability of individual agents, shared goals have not yet been implemented; the user sets a goal for the system, and the system carries it out. This is not a realistic assumption, and in future work we expect to extend ICF to better represent task failure and repair. In the long term, a more complete implementation of teamwork will be necessary.

Assessment of Deontics Implementation

Although the deontics implementation adds value as described above, there are also unresolved issues. For example, we lack a general solution to deontic operator conflicts, such as when a prohibition affects the same goal as an obligation. These conflicts require a conflict resolution mechanism involving some method of judging deontic operator utility. A general utility measure for real-world performance of a military team is not easy to come by, however. Many metrics could be chosen to fill this purpose (optimizing for minimal losses, for maximal disabling of the enemy, for minimal time, etc). We have found none that is correct for every mission, and believe no fixed metric could be correct. We implemented a limited "most recent" conflict resolution mechanism, which resolves a conflict between two deontic operators by giving preference to the one that was generated most recently.

The deontic obligation operator has proven to be an inherently challenging one, in that it may require an agent to generate a task for which it does not have fully specified information. For each of the deontic operators, a characterization of the goal or transform is enough to define whether a deontic feature affects an existing goal or transform, or not. This is sufficient information to ensure that a prohibition, a permission-with-permission, or a permission deontic operator is properly followed. However, when dealing with an obligation, the agent is expected to ensure that a suitable goal or transform is eventually adopted. If an obligation is fully grounded or affects an already-existing transform or goal (e.g., an agent has already been planning an attack), then it is relatively straightforward to create and/or make that activation mandatory. However, generation of a fully specified goal or transform based on an underspecified obligation is currently beyond the system's capabilities. ICF does not

yet include a planning system that could fill in the details and transform an underspecified obligation into a fully specified obligation.

The form of the deontic operators entails another interesting challenge with respect to obligations. In the operator specification, there is no information about when an obligation must be fulfilled. This can lead to a situation where an agent, making a decision at time *t*, chooses to put off fulfilling the obligation until time *t+1*. Then, at time *t+1*, the agent may again choose to put off fulfilling the obligation. Thus, the agent may choose to *never* fulfill the obligation, violating the very nature of obligation. One way to ensure execution that meets the obligation might be to always specify a temporal constraint as part of the goal description, such as a deadline.

Agent Communication

ICF agents coordinate using a message-passing paradigm. We have found that ad-hoc message passing in a large system can lead to inconsistent, ambiguous, and ultimately confusing communication semantics. In addition, there are naturally occurring messaging patterns (e.g., communication protocols) that are difficult to implement efficiently in an ad-hoc messaging system

Because of these difficulties, we decided to utilize a formal Agent Communication Language (ACL) paradigm for inter-component messages. We chose an ACL based on JIT [23]. This ACL defines a broad suite of message types and protocols; we have implemented *inform* (convey beliefs), *request* (ask for action from a party), *shout* (multi-cast INFORM), *subscribe* (standing REQUEST for information updates), and *query* (one-time REQUEST for information).

Assessment of ACL Implementation

The ACL provides significant benefits to the JIT-based, multi-agent teamwork model described above. It has also allowed standardization of some message content, generalization of some message protocols and effects (e.g., an accepted request leads to some agent action), and enables the agent to track conversations over the course of several messages.

While the ACL is valuable for general, robust communication, it has introduced several implementation challenges. In particular, there is a ubiquitous term in the formal logical definitions that permits specification of when the JIT goal-related expressions are no longer relevant. Traditionally, these "relativizing conditions" contain an expression related to such things as validity of a higher-level goal or situational conditions that obviate particular JIT expressions [9]. Because of the generality of the JIT formalism, there is little guidance as to the form the relativizing condition should take in any particular implementation. Because JIT expressions are conveyed between system components, the relativizing condition will be encoded by a different component than the component that evaluates it. As in any knowledge sharing situation, this can be problematic without implementation safeguards (e.g., design conformity, ontological support, agent capability registration, and compatibility matching).

An additional issue we faced was that the ACL approach requires the implementation of ACL "wrappers" for non-agent components in the system. The wrapper allows these components to properly fill message templates and play their roles in the conversation protocols. While this requirement has the positive effect of enforcing consistent communication across all components, it also adds unnecessary overhead to many messages.

Conclusion

While the implementation of intention-related formalisms for ICF is incomplete, the portion of those formalisms that we have implemented has enabled a capable and flexible system. We have deliberately selected those parts of the formalisms that address immediate needs and ignored parts not yet needed. In this way, we have avoided costly over-engineering and extra development that might never have contributed to ICF's primary goals. As an additional benefit, because the formalisms are internally consistent, they provide potential for future growth in agent capabilities. This approach allows us to progressively extend the implemented formalisms when and if we need unimplemented elements, with a minimum of incompatibilities and surprises. The advantage comes particularly from the use of templates and partitioning of agent code by subject goal or transform. This contrasts with Soar-only models, in which extensions can often interfere in surprising and difficult-to-decipher ways as they compete for space in the operator stack. Thus, the formalisms of intention are an excellent basis for design of an agent system, but it is important to carefully choose which parts of the formalisms will be included in a specific engineering effort.

Additionally, we have found that intentional gaps in the implementation of a formalism or theory can leave useful flexibility. An example is Soar's lack of commitment to particular structures within working memory, which allowed the addition of the BDI abstraction layer in the form of a library. Least-commitment is not only a useful strategy for artificially intelligent agents, it can also be quite useful for the naturally intelligent scientists, programmers, and systems architects who work with them.

Acknowledgements

We thank our sponsors, Drs. Greg Hudas and Jim Overholt of U.S. Army RDECOM-TARDEC, and our collaborators at Soar Technology who helped conceptualize the BDI abstraction layer, including Jon Beard, Jacob Crossman, Randolph Jones, and Glenn Taylor.

References

[1] J. E. Laird, A. Newell, and P. S. Rosenbloom, "Soar: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, pp. 1-64, 1987.

[2] D. Morley and K. Myers, "The SPARK agent framework," presented at Autonomous Agents and Multiagent Systems Conference (AAMAS 2004), 2004.

[3] R. M. Jones, J. E. Laird, P. E. Nielsen, K. J. Coulter, P. G. Kenny, and F. V. Koss, "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, vol. 20, pp. 27-42, 1999.

[4] R. E. Wray, J. E. Laird, A. Nuxol, D. Stokes, and A. Kerfoot, "Synthetic Adversaries for Urban Combat Training," in *AI Magazine*, vol. 26. San Jose, CA, 2005, pp. 82-92.

[5] R. E. Wray and R. M. Jones, "Considering Soar as an Agent Architecture," in *Cognition and Multi-agent Interaction: From Cognitive Modeling to Social Simulation*, R. Sun, Ed. Cambridge, UK: Cambridge University Press, 2005.

[6] R. B. Doorenbos, "Combining left and right unlinking for matching a large number of learned rules," presented at Twelfth National Conference on Artificial Intelligence (AAAI-94), Seattle, Washington, 1994.

[7] H. Pashler, "Dual-task inteference in simple tasks: Data and theory," *Psychological Bulletin*, vol. 116, 1994.

[8] J. Thangarajah, L. Padgham, and J. Harland, "Representation and Reasoning for goals in BDI agents," presented at Proceedings of the twenty-fifth Australasian conference on Computer Science - Volume 4, Melbourne, 2002.

[9] P. R. Cohen and H. J. Levesque, "Performatives in a rationally based speech act theory," presented at Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics, 1990.

[10] M. Tambe, "Agent architectures for flexible, practical teamwork," *National Conference on AI (AAAI97)*, pp. 22--28, 1997.

[11] G. Taylor, R. M. Jones, M. Goldstein, R. Fredericksen, and R. E. Wray, "VISTA: A Generic Toolkit for Agent Visualization," presented at 11th Conference on Computer Generated Forces and Behavioral Representation, Orlando, 2002.

[12] G. Taylor, R. Frederiksen, R. R. Vane, and E. Waltz, "Agent-based Simulation of Geo-Political Conflict," presented at 2004 Innovative Applications of Artificial Intelligence Conference, San Jose, CA, 2004.

[13] L. Braubach, A. Pokahr, W. Lamersdorf, and D. Moldt, "Goal Representation for BDI Agent Systems," presented at Second International Workshop on Programming Multiagent Systems: Languages and Tools, 2004.

[14] R. M. Jones and R. E. Wray, "Comparative Analysis of Frameworks for Knowledge-Intensive Intelligent Agents," in *AI Magazine*, vol. 27, 2006, pp. 57-70.

[15] Y. Lallement and B. E. John, "Cognitive architecture and modeling idiom: A model of the Wicken's task," presented at Twentieth Annual Conference of the Cognitive Science Society, Madison, Wisconsin, 1998.

[16] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng, "Distributed intelligent agents," *IEEE Expert*, vol. 11, pp. 36-46, 1996.

[17] R. E. Wray and J. E. Laird, "An architectural approach to consistency in hierarchical execution," *Journal of Artificial Intelligence Research*, vol. 19, pp. 355-398, 2003.

[18] J. E. Laird and R. M. Jones, "Building Advanced Autonomous AI systems for Large Scale Real Time Simulations," presented at Proceedings of the 1998 Computer Game Developers' Conference, Long Beach, Calif., 1998.

[19] M. Georgeff and A. L. Lansky, "Reactive Reasoning and Planning," presented at Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, Washington, 1987.

[20] M. J. Huber, "JAM: A BDI-theoretic Mobile Agent Architecture," presented at Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA, 1999.

[21] M. Tambe, A. Newell, and P. S. Rosenbloom, "The problem of expensive chunks and its solution by restricting expressiveness," *Machine Learning*, vol. 5, pp. 299-348, 1990.

[22] P. R. Cohen and H. J. Levesque, "Teamwork," *Nous*, vol. 35, 1991.

[23] M. J. Huber, S. Kumar, and D. McGee., "Toward a Suite of Performatives based upon Joint Intention Theory," presented at Proceedings of the AAMAS 2004 Workshop on Agent Communication, New York, NY, 2004.

[24] S. A. Lisse, J. T. Beard, M. H. Huber, G. P. Morgan, and E. A. M. DeKoven, "A deontic implementation of adjustable autonomy for command and control of robotic assets," *Proc. SPIE Int. Soc. Opt. Eng*, vol. 6230, pp. 62300C, 2006.