

Comparative Analysis of Frameworks for Knowledge-Intensive Intelligent Agents

Randolph M. Jones and Robert E. Wray

■ A recurring requirement for human-level artificial intelligence is the incorporation of vast amounts of knowledge into a software agent that can use the knowledge in an efficient and organized fashion. This article discusses representations and processes for agents and behavior models that integrate large, diverse knowledge stores, are long-lived, and exhibit high degrees of competence and flexibility while interacting with complex environments. There are many different approaches to building such agents, and understanding the important commonalities and differences between approaches is often difficult. We introduce a new approach to comparing frameworks based on the notions of commitment, reconsideration, and a categorization of representations and processes. We review four agent frameworks, concentrating on the major representations and processes each directly supports. By organizing the approaches according to a common nomenclature, the analysis highlights points of similarity and difference and suggests directions for integrating and unifying disparate approaches and for incorporating research results from one framework into alternatives.

Overview

One frequently taken approach toward achieving human-level intelligent systems is to create foundational software systems that tightly integrate some number of representations and processes deemed suffi-

cient for generating automated intelligent behavior. The design of these foundational software systems, which include both cognitive and agent architectures, have generally been based on some small set of theoretical principles. The agent architecture is an attempt to foster the development of uniform approaches for building intelligent systems. However, large-scale integrated software systems that attempt to approach human levels of intelligence through agent architectures exhibit some core commonalities across different architectures. For example, no matter the chosen architecture, there is a necessity for such systems to encode vast amounts of knowledge in efficient, organized, and maintainable ways. Additionally, these knowledge requirements have had relatively uniform effects on the evolution of these architectures, such that we observe a convergence of essential representations and processes across agent architectures.

A variety of frameworks currently exist for designing human-level intelligent agents and behavior models. Although they have different emphases, each of these frameworks provides coherent, high-level views of intelligent agency. However, more pragmatically, much of the complexity of building intelligent agents occurs in the low-level details, especially when building agents that exhibit high degrees of

competence while interacting in complex environments. To highlight the emphasis of our observations about the knowledge necessary for human-level artificial intelligence, we call such agents *knowledge-intensive agents*. This term is also meant to distinguish such agents from smaller-scale, single-task agents (for example, service brokers) that are often fielded in multi-agent systems. Examples of fielded knowledge-intensive agents include a real-time fault diagnosis system on the Space Shuttle (Georgeff and Ingrand 1990) and a real-time model of combat pilots (Jones, Laird, and Nielsen 1999). Knowledge-intensive agents are also often used in “long-life” situations, where a particular agent needs to behave appropriately and maintain awareness of its environment for a long period of time (hours to days) while performing many different activities during the span of its existence. Additionally, knowledge-intensive agents must be engineered such that their knowledge can be easily modified (possibly by both extrinsic and intrinsic processes) as environment and task requirements change during deployment.

Transfer and generalization of results from one framework to others is usually slow and limited. The reasons for such limited transfer include differences in nomenclature and methodology that make it more difficult to understand and apply results, and the necessity of specifying low-level details that are not prescribed by the frameworks but that become important in actual implementation. In addition, high-level agent frameworks do not usually guide the agent developer in many finer-grained implementation issues, meaning that the frameworks underspecify necessary principles to build and field working intelligent agents. Our goal is to develop techniques that will minimize framework-specific descriptions and that bridge the gap between a framework’s theory and the details of its implementation, especially clarifying which details are intrinsic to particular approaches and which are not. In the long run, this effort should foster reuse of architectural components and idioms across architectures as well as across individual agent models that use a single architecture.

This article reviews four existing agent frameworks in order to explore what they specify (and do not) about an agent’s design and construction. The chosen frameworks have proven successful for building knowledge-intensive agents of various levels of complexity, or specifically address constraints on agents with high levels of competence (such as human behavior models). We identify the representations and agent processes that the frameworks

dictate for agent design. This comparative analysis, to our knowledge, is novel and provides insights into the trade-offs inherent in these systems for building intelligent agents. The goal is truly comparative. Each system we review arguably has a unique application niche, and we are not seeking to suggest one framework is better than another. Rather, in comparing them, especially in noting convergences and divergences in knowledge-intensive agent applications, we seek to develop a uniform methodology for comparing frameworks and, ultimately, to speed the development and evolution of architectures by making research results more communicable and transparent to researchers not working within the specific subfield of AI or cognitive science in which new architecture developments are made.

One important result of this analysis is the observation that no single framework we review here directly supports all of the representations that have been usefully employed in various knowledge-intensive agent systems. The result is that an agent designer who adopts any of these particular frameworks often must also develop application-specific solutions for the representations and processes not directly supported by the chosen framework. This situation is undesirable because it leads to ad hoc solutions for different agent applications created within the same framework. Ad hoc solutions in turn increase development costs by hampering reuse. While the current analysis does not provide a complete set of necessary representations and processes for knowledge-intensive agents, it does serve as a starting point for future architectural research: creating and deploying robust, lower-cost, long-lived agent applications makes it essential to have direct architectural support of all the basic representations and processes.

Review of Agent Frameworks

We introduce four mature frameworks for intelligent agents that represent quite different theoretical traditions (philosophical and logical, functional, psychological, and formal computational). We have intentionally selected exemplar frameworks that are somewhat different in character in order to provide a broad first cut at an encompassing review. Our intent is to consider the primary representational constructs and processes directly supported by each. We focus on these aspects of agent frameworks because an agent is essentially the sum of a system’s knowledge (represented with particular constructs) and the processes that operate on those constructs (Russell and Norvig 1994).

We focus on frameworks that have been used to build large-scale, highly capable agent systems because different programming paradigms are likely appropriate for systems with leaner knowledge. An example motivating factor for this analysis is the recognition that *implemented* BDI and Soar systems, while originating from different theoretical starting points, have converged on similar solutions for large-scale systems. However, this analysis can be extended to other frameworks as well, with still other representations and processes (for example, 4D/RCS [Albus 2001], ACT-R [Anderson and Lebiere 1998], Icarus [Langley, Choi, and Shapiro 2004], and RETSINA [Payne, Singh, and Sycara 2002]). In the long term, we will extend our analysis to these other frameworks as well.

BDI

The BDI (beliefs, desires, intentions) framework grew out of Bratman's (1987) theory of human practical reasoning. BDI is now a popular logic-based methodology for building competent agents (Georgeff and Lansky 1987; Rao and Georgeff 1995; Wooldridge 2000). A basic assumption in BDI is that intelligent agents ought to be rational in a formal sense, meaning rationality (as well as other properties) can be logically proven. Actions arise from internal constructs called intentions. An intelligent agent cannot make decisions about intentions until it has at least some representation of its beliefs about its situation. That is, the agent must maintain a set of beliefs about what is true in the world. Given a particular set of beliefs, there may be many different situations that the agent might consider desirable. Given limited resources, however, the agent can often only act on some subset of these desires, so the agent selects a subset, its intentions, to pursue. Using BDI terminology, the entire set of relevant activities represents the agent's *desires*, and the set of currently selected actions that address some subset of those desires are the *intentions*.

BDI was also designed with specific high-level constraints on intelligent behavior in mind. First, as mentioned, the framework insists on rational agents, in the sense that a BDI agent's actions must always be logically consistent with its combination of beliefs and goals. This property is not true of some of the other frameworks we analyze, particularly those with a heavy emphasis on psychology (where intelligent behavior that is not strictly rational is observed with some frequency). Second, the BDI framework also emphasizes supporting groups of agents that interact with each other. BDI is a

high-level framework that has a number of distinct implementations, among them IRMA (Bratman, Israel, and Pollack 1988), PRS (Georgeff and Lansky 1987), dMARS (d'Inverno et al. 1997), JACK (Howden et al. 2001) and JAM (Huber 1999). Our discussion includes some small examples of differences in implemented architectures where those architectures have made specific commitments beyond the general BDI framework. However, in general our consideration of BDI is meant to be consistent with the common framework as presented by Wooldridge (2000).

GOMS

GOMS (goals, operators, methods, and selections) is a methodology based in psychology and human-computer interaction (Card, Moran, and Newell 1983). GOMS is not strictly an agent framework, but it formalizes many details of high-level human reasoning and interaction. However, GOMS is particularly interesting because knowledge-intensive agents are often used to simulate human behavior. Although GOMS has not been used to develop large-scale systems, it has been used to represent the human knowledge necessary for performing many tasks, including complex human activity. We include GOMS because the representation and process regularities it has identified are critical for knowledge-intensive agents that will encode this type of knowledge. In addition, improvements in efficiency increasingly allow executable cognitive models to compete with AI architectures in application areas (for example, John, Vera, and Newell [1994]).

GOMS systems explicitly encode hierarchical task decompositions, starting with a top-level task goal plus a number of methods, or plans, for achieving various types of goals and subgoals. Each goal's plan specifies a series of actions (called operators by the GOMS community) invoking subgoals or primitive actions to complete the goal. Selection rules provide conditional logic for choosing between plans based on the agent's current set of beliefs.

One key feature of GOMS is its support for hierarchical task decomposition. Although a hierarchical model is not a strict requirement, among the frameworks examined here GOMS most strongly encourages and supports hierarchical solutions. Like BDI, GOMS is a high-level framework, realized in a number of individual implementations, such as GLEAN (Kieras et al. 1995), APEX (Freed and Remington 2000), CPM-GOMS (Gray, John, and Atwood 1993), and NGOMSG (Kieras 1997).

Soar

Soar has roots in cognitive psychology and computer science, but it is primarily a functional approach to encoding intelligent behavior (Laird, Newell, and Rosenbloom 1987). The continuing thread in Soar research has been to find a minimal but sufficient set of mechanisms for producing intelligent behavior. These goals have resulted in uniform representations of beliefs and knowledge, fixed mechanisms for learning and intention selection, and methods for integrating and interleaving all reasoning.

Like BDI, Soar's principles are based in part on assumed high-level constraints on intelligent behavior. Foremost among these are the problem space hypothesis (Newell 1982) and the physical symbol systems hypothesis (Newell 1980). Problem spaces modularize long-term knowledge so that it can be brought to bear in a goal-directed series of discrete steps (on the surface, this modularization is somewhat similar to the encapsulation of actions provided by FSMs, described later). The problem space hypothesis assumes rationality, similar to BDI. The physical symbol systems hypothesis argues that any entity that exhibits intelligence can be viewed as the physical realization of a formal symbol-processing system. The physical symbol systems hypothesis led to Soar's commitment to uniform representations of knowledge and beliefs.

There is no explicit assumption of hierarchical task representations in Soar (as there is in GOMS), but in practice the use of problem spaces often leads to the development of hierarchically organized behavior models, in which each portion of the hierarchy may represent a different problem space. However, the general notion of problem spaces also supports other types of goal arrangements and context switching.

While Soar shares with BDI the notion of agent rationality (agents appropriately select actions in pursuit of goals) and Soar uses logic-based knowledge representation, Soar does not share BDI's commitment to logical reasoning to produce rational behavior. Thus, Soar imposes strong constraints on fundamental aspects of intelligence, but it does not impose functionally inspired high-level constraints (in the spirit of BDI's use of logic, or GOMS's use of hierarchical goal decomposition). Soar is a lower-level framework for reasoning than BDI and GOMS. Either BDI principles (Wray and Jones 2005) or GOMS principles (Peck and John 1992) can be followed when using Soar as the implementation architecture.

FSMs

FSM (finite state machine) approaches to intelligent agents come from theoretical computer science (Carmel and Markovitch 1996; Hopcroft and Ullman 1979). Their primary appeal is the simplicity of their representational elements, which can usually be easily understood and encoded, automatically learned for some tasks, and implemented very efficiently. However, it is worth noting that some of these advantages diminish for models of large size or high complexity. Because states and transitions are relatively simple and low level, FSMs do not present the same types of high-level architectural constraints as the other frameworks we review here. Rather they provide theoretically sound elements from which more complex systems can be built. FSMs achieve complexity in behavior by the complex design of states and transitions. Because of the relatively simple level of support for knowledge representation idioms, some would argue that FSMs do not represent a knowledge-intensive agent framework at all. However, FSMs are used for a variety of agent applications, especially in computer games and human behavior representation (Ceranowicz, Nielsen, and Koss, 2000), so it is worth considering this approach in our analysis.

In the purest form of FSM, the only representational commitment is the state itself, which uniquely represents some point in the space of all possible combinations of beliefs and goals. While this commitment may seem minimal in comparison to the other frameworks, in practice FSMs provide additional ways to implement many of the constructs shared by other agent frameworks. For example, FSMs are functionally equivalent to a set of propositional stimulus-response rules, in which the state uniquely determines an agent's action, given its knowledge base. In practice, however, it is just as difficult to build a knowledge-intensive system using pure FSMs as it would be to use a purely propositional set of rules. Thus, practical FSM systems extend the approach by, for example, supporting variables within and across states, allowing conditional execution, and in some cases providing a global memory store.

Like BDI and GOMS, FSMs provide a general framework that has been implemented in a wide variety of systems. Because FSMs can be easily implemented within standard procedural programming languages, they are often equipped with additional features that violate the strict FSM paradigm. For example, FSMs can be hierarchically combined to allow multiple goals and task decompositions. In such an implementation, entering a state in one ma-

	Representation	Commitment	Reconsideration
Inputs			
<i>BDI</i>	Input language		
<i>GOMS</i>	Input language		
<i>Soar</i>	Working memory		
<i>FSM</i>	State transitions		
Justified Beliefs			
<i>BDI</i>	Beliefs	Logical inference	Belief revision
<i>GOMS</i>	Working memory		
<i>Soar</i>	Working memory	Match/assert	Reason maintenance
<i>FSM</i>	State variables		
Assumptions			
<i>BDI</i>	Beliefs	Plan language	Plan language
<i>GOMS</i>	Working memory	Operators	Operators
<i>Soar</i>	Working memory	Deliberation/Ops	Operators
<i>FSM</i>	State variables	Assignment	Assignment
Desires			
<i>BDI</i>	Desires	Logic	Logic
<i>GOMS</i>			
<i>Soar</i>	Proposed ops.	Preferences	Preferences
<i>FSM</i>			
Active Goals			
<i>BDI</i>	Intentions	Deliberation	Decision theory
<i>GOMS</i>	Goals	Operators	
<i>Soar</i>	Beliefs/Impasses	Deliberation	Reason maintenance
<i>FSM</i>	State machine	Context switching	Context switching
Plans			
<i>BDI</i>	Plans	Plan selection	Soundness
<i>GOMS</i>	Methods	Selection	
<i>Soar</i>			Interleaving
<i>FSM</i>	Transition networks	Context switching	
Actions			
<i>BDI</i>	Plan language	Atomic actions	
<i>GOMS</i>	Operators	Operators	
<i>Soar</i>	Primitive Ops	Deliberation	Reason maintenance
<i>FSM</i>	State transitions	Serial control flow	
Outputs			
<i>BDI</i>	Plan language	Plan language	
<i>GOMS</i>	Primitive ops.	Conditional ops.	
<i>Soar</i>	Working memory	Conditional ops.	
<i>FSM</i>	Output transitions	Serial control flow	

Table 1. Agent Framework Comparisons.

Black items are specific solutions provided by the framework. Gray items are general support provided by the framework. No entry means the framework does not explicitly address the element.

chine causes a jump into the initial state of another machine (with a subsequent jump back when the second machine completes execution).

Analysis of Agent Frameworks

Each of these frameworks provides a coherent view of agency and gives explicit attention to specific representations and processes for intelligent agents. They also reflect different points of emphasis, arising in part from the theoretical traditions that produced them. However, because none of the frameworks cover all the points of emphasis, agent designers have to make many more decisions about agent construction than provided by each framework's core principles. Each architectural implementation requires nonprimitive representations (and the processes to manipulate these representations). While it is likely that an agent will have compositional representations (for example, a representation of a map composed of an indexed set of beliefs), general representational constructs that span most applications should be directly supported within the framework.

Direct support simplifies the development process because the agent designer can concentrate exclusively on the domain knowledge. The practical point of an agent framework is to provide a set of reusable elements in order to reduce the costs of building new agents. One could therefore argue that, to maximize reuse, any representational element that is general across domains and useful in the majority of agent applications should be required to be addressed in each framework. However, this desire for reusability must be taken in context with the additional functional and theoretical constraints associated with each framework. For our analysis, we will comprehensively list each representational element supported by any of the frameworks, and note where individual frameworks provide support (or not) for those elements.

Table 1 lists the union of the base-level representations from BDI, GOMS, Soar, and FSMs. The representations are ordered to suggest the basic information flow from an external world into agent reasoning and then back out. The "Representation" column specifies each framework's substrate for the base-level representational element. Each representation also requires a decision point in the reasoning cycle, where an agent must choose from a set of alternatives. We generalize Wooldridge's (2000) concept of intention commitment to specify the process an agent uses to assert some instance of the base-level representation. In Table

1, the "Commitment" column identifies the general process used to select among alternatives. Most commitments also require maintenance; the "Reconsideration" column shows the determination of whether a commitment remains valid (a generalization of the notion of intention reconsideration [Schutt and Wooldridge 2001]).

Perceptions

Any interactive agent must have a perceptual or input system that provides a primitive representation of the agent's environment or situation. Neither BDI nor GOMS specifies any particular constraints on input. FSMs generally specify input conditions for initial states, as well as for state transitions. These conditions usually correspond to percepts in the environment, but the FSM approach makes no commitment to the specific representation of these conditions or to their grain size. Soar represents primitive perceptual elements in the same attribute-value representation as beliefs, although it does not dictate the structure of the perceptual systems that create these elements. However, the constraint that perceptual input must be represented in the same language as beliefs has important implications. Aside from their location in memory, primitive perceptual representations are indistinguishable from beliefs, which is consistent with Soar's principle of uniform knowledge representation. This makes it a relatively simple matter to allow agents to deliberate over potential input situations (or reflect on past or possible future input experiences) and transfer that knowledge directly to actual inputs.

Beliefs

From primitive perceptual elements, an agent creates a further elaborated set of beliefs, or interpretations of the environment. The set of beliefs is sometimes referred to as the *current state* of the agent, a concept that is made explicit in FSMs. Using the terminology of reason maintenance systems (Forbus and deKleer 1993), beliefs can be classified as either justified beliefs or assumptions. Justified beliefs remain in memory only as long as they are logically entailed by perceptual representations and assumptions. Assumptions, by definition, remain in memory independently of their continuing relevance to—and logical consistency with—the external environment. Assumptions remain asserted until the agent explicitly removes them, with the result that assumptions receive a high degree of commitment from the agent. Assumptions are necessary because not all beliefs can be grounded in current perception. For exam-

ple, if an agent needs to remember an object no longer in the field of view, then it must commit to maintaining a memory of that object. As long as the object remains in the field of view, the agent's perception of the object can be considered a *justified belief* (sometimes called an *entailment*). As soon as the perceptual grounding disappears, the agent must commit to the belief as an *assumption* if it is going to maintain the belief for some time.

Neither GOMS nor BDI makes an explicit distinction between justified beliefs and assumptions. Each provides general mechanisms for maintaining justified beliefs, but not specific solutions. Belief revision (Gardenfors 1988) is the mechanism of justified belief reconsideration in the BDI framework, although details of the process are only defined in various specific implementations. Soar uses a reason maintenance system to assert and retract justified beliefs automatically. Reason maintenance ensures that justified beliefs are logically consistent (Wray and Laird 2003). All four frameworks support the representation of assumptions. Soar requires that assumptions be created as the result of deliberate commitments (operator effects).

Pure FSMs would only be allowed to represent combinations of beliefs with individual states, because they are prohibited from maintaining internal state information. However, this would lead to an enormous and unmanageably complex set of states. Probably for this reason, we are not aware of any practical agents that are implemented using pure FSMs. Rather, the machines are generally augmented with variables that can hold various types of "non-state" information. Variable values represent assumptions, because no primitive process maintains the continuing validity of a value with respect to the external situation.

Importantly, other frameworks use still other techniques for managing the commitment and reconsideration of beliefs. For example, 4D/RCS (Albus 2001) uses a limited capacity buffer, allowing only a fixed number of assumptions to be asserted at any one time. ACT-R (Anderson and Lebiere 1998) employs sub-symbolic activation and decay mechanisms to manage assertions. By making such design decisions explicit in this analysis, we hope to facilitate a discussion of the trade-offs in these decisions among different approaches, and to make it more clear how to incorporate mechanisms from one architecture to another. For example, the activation and decay mechanisms of ACT-R have recently been incorporated into a hybrid architecture integrating Elements of ACT-R, Soar, and EPIC (EASE) (Chong and Wray

2005). EASE uses Soar's reason maintenance system to manage the assertion and retraction of justified beliefs, but uses ACT-R's activation and decay mechanisms to manage assumptions. These alternative belief representations do not follow strict logical entailment, but also do not require deliberate agent reconsideration, so it is likely that we should include other types of beliefs as our analysis progresses. One of the contributions of this work is to provide a formal theoretical framework in which such variations in belief commitment and reconsideration can be labeled and characterized.

Desires

BDI is the only framework that clearly separates desires from "normal" active goals (below). Desires allow an agent to monitor goals that it has chosen not to pursue explicitly. An additional advantage is that an agent can communicate its desires to another agent that may be able to achieve them (Wooldridge 2000). Even in single-agent applications, however, there may be situations where an agent would need to reason about a desire, even if it does not have the resources to pursue that desire. Such situations may provide the possibility of opportunistically achieving desires in the context of other active goals.

Unlike BDI, agents built within many other frameworks do not bother to represent goals that they do not intend to pursue. Soar, GOMS, and FSMs do not specify that desires should exist, how they should be represented, or how they should influence reasoning. In these agents, expressing a desire would consist of a deliberate act in the service of a communication goal. BDI manages commitment to desires through logical inference.

Active Goals

A hallmark of intelligent behavior is the ability to commit to a particular set of concerns and then pursue them (Bratman 1987; Newell 1990). Most agent frameworks support explicit representation of the goals an agent has committed to pursue. However, the agent literature is somewhat inconsistent in its use of descriptive terms relevant to goals, which is a continuing source of confusion and miscommunication in the research community. Wooldridge (2000) calls *active goals* "intentions." In contrast, some implementations of BDI do not represent active goals distinctly from the selected plans that would achieve these goals. In such systems, *selected plans* are "intentions," but there is no explicit representation of an active goal apart from the plan. In Soar, an "intention" is the *next action selected* from a current

plan (which may itself directly activate a goal). GOMS does not use the term “intention,” but requires the explicit representation of goals. In an attempt to avoid confusion, we call these commitments “active goals” to distinguish them from plans and (“inactive”) desires. We also avoid altogether the ambiguous and overloaded term “intention.”

Agents require a process for selecting the current active goal (or set of goals). BDI and Soar include explicit processes for deliberate goal commitment, although goals can be implemented in a variety of ways in Soar. In particular, goals created directly by the Soar architecture are limited to impasse goals; that is, goals to solve a particular problem in execution. While some approaches to Soar map task goals (for example, “intercept the aircraft”) to impasse goals, this approach is only one of a number of “idioms” that are used to represent goals within Soar models (Lallement and John 1998). In GOMS, goal commitment occurs by invoking the plan associated with the goal. Although this is a deliberative process, it is not divided into separate steps as in the other frameworks. FSMs do not have an explicit notion of active goals. Implicitly, each FSM represents a plan that is associated with a particular goal (or set of goals).

Researchers have also explored the question of when an agent should reconsider an active goal (for example, Veloso, Pollack, and Cox [1998]; Schutt and Wooldridge [2001]; Wray and Laird [2003]). The BDI framework uses evaluations of soundness to determine when an active goal should be reconsidered; that is, given the agent’s beliefs, the plan provably achieves the active goal. More recently, BDI researchers have also explored decision-theoretic processes for intention reconsideration (Schutt and Wooldridge 2001). Soar utilizes reason maintenance, which is essentially an implementation of the soundness criterion. GOMS uses selection rules to commit to a goal, but does not explicitly address later reconsidering a goal. An FSM would normally mark one or more of its states as states that achieve some (implicit) goal, perhaps terminating an individual state machine when a goal is achieved (although this approach would be different for *maintenance* goals).

Plans

Once there is an active goal to pursue, the agent must commit to a plan of action. BDI and GOMS assume there is a plan library or some other method for generating plans outside the basic agent framework (GOMS includes the notion of methods but does not prescribe how

methods are implemented). Generally, plan execution is implicit in FSMs, so there is no explicit representation of finding a plan to achieve the goal. Each individual state machine is a plan, using states and transitions to capture the execution of the plan in the service of some (usually implicit) goal.

Soar does not require that an agent have an explicit representation of a plan. More commonly, Soar agents associate individual actions directly with goals (plan knowledge is implicit in the execution knowledge), or interleave planning and execution as individual cognitive tasks. Either way, Soar assumes that planning is a deliberative task requiring the same machinery as any other agent activity and involving the same concerns of commitment and resource usage. However, as with any other unsupported base-level representation, Soar forces the agent developer to implement the planning algorithm and the representation of any plans. Alternatively, external planning tools can be used to generate plans that must then be converted into Soar’s belief representation language or production rules.

GOMS and BDI do not specify plan languages, although their implementations do. Soar has nothing like the relatively rich GOMS and BDI plan languages, instead using its operators to implement simple types of commitment. The trade-off is that complex plans are easier for a developer to program in GOMS and BDI, but potentially easier to learn by a Soar agent (because of the simpler, uniform target language). Developers of GOMS and BDI implementations must make decisions about plan languages, leading to nonuniform solutions from one implementation to another. The “plan language” for an FSM is generally just the same computer language used to implement the FSM.

Plan commitment in BDI can be quite simple: plans can be selected through a lookup table indexed by goal (Wooldridge 2000) or implied completely by goal selection, as in JAM. In sharp contrast, GOMS treats the choice of which method to choose to pursue a goal as a major element of the framework. Because Soar does not have an architectural notion of a plan, there is no plan-specific commitment mechanism. Soar also does not make an explicit distinction between plan generation/selection and plan execution. Creating (or finding) a plan involves a series of context-sensitive decisions and operations, just as executing a plan does.

An agent can consider abandoning its current plan, even when it has chosen to remain committed to its current goal (Wooldridge

2000). The frameworks here do not provide strong advice on when such a commitment should be given up; BDI and Soar at least dictate that any plan should be executed one action at a time, allowing reconsideration of the plan after each step (although the two disagree on how complex a single action can be). Frameworks that use explicit plans may provide support for abandoning a plan reactively (BDI) or ignore this problem completely (GOMS). Soar, because it does not require explicit plans, implicitly supports plan reconsideration, because there is no separate commitment to a plan in the first place. Thus, in Soar, an agent commits to one action at a time rather than committing to a whole plan. This embodies a strong *least-commitment* approach to plan selection and execution in general. The trade-off is that Soar agents must include extra knowledge to remain committed to a particular course of action, and the implementation of this knowledge is up to individual agent developers.

Other approaches to plan maintenance include using *completable plans* (Gervasio and DeJong 1994) and allowing agents to switch back and forth between two or more plans in support of multiple, orthogonal goals. Completable plans are plans that specify behavior to some abstract (but relatively low) level, and then allow the abstractions to instantiate conditionally and reactively to changing environments during execution. Plan switching is clearly a requirement for knowledge-intensive agents in many complex domains, but none of the frameworks specify how switching must occur. For example, it is not clear that any current implementations of BDI or GOMS support resumption of a partially executed plan. Many Soar systems implement task switching, but they rely on extra knowledge coded by the agent developer. Such *reentrant* execution of plans appears to be an essential element of opportunistic intelligent behavior.

Actions

Regardless of whether a plan has been explicitly represented, an agent must eventually commit to some type of action relevant to its active goals. In frameworks with explicit plans, like BDI and GOMS, this involves following and executing the steps in the plan. Explicit actions in FSMs simply involve moving from one state to another (or in augmented FSMs, possibly executing some code that changes the belief set or issues output commands). At their core, all four frameworks support three general types of actions: execute an output command, update the belief set, or commit to a new goal (or desire). GOMS and Soar define operators as the atomic

level of action, allowing commitment and reconsideration for each plan action. As an alternative, BDI and FSM systems generally provide a plan language that is a complete programming language. Such languages provide powerful and flexible means of plan implementation, but may leave them outside the commitment regime of the framework. BDI dictates that reconsideration ought to occur after each plan step, but does not tightly constrain how much processing may occur in a single step. This imposes a trade-off between ease of programming (BDI and FSMs) and taking advantage of the uniformity of the framework's built-in processes (GOMS and Soar).

Soar uses actions to create assumptions in the belief set (thus, assumptions can only be the result of deliberate decision making). Tying assumptions to actions is an important issue. Automated, logical reason maintenance is attractive, but, pragmatically, there are limited resources for updating an agent's beliefs. Ideally, a rational agent would compute all relevant entailments from any input. But in complex environments, this is not computationally feasible (for example, Hill [1999]).

Regardless of the particular approach to plan representation or action languages, all the agent frameworks represent an action as a discrete step in a current plan's pursuit of a goal. If it happens to be a discrete step in an abstract plan, then it may get further decomposed (completable planning). In addition, each framework generally initiates a discrete action every "tick of the clock." This is how agents make progress towards their goals, and it allows a commitment scheme where reconsideration (of plans, goals, or beliefs, depending on the agent) can occur after each discrete action.

Outputs

The ultimate level of commitment is to initiate activity in the environment. To accomplish this, an agent invokes an output system. All four frameworks assume that output has to happen somehow, but do not impose strong constraints on the representation of output. BDI leaves output decisions up to the designer of the plan language. GOMS requires that primitive operators produce all output signals. As with perception, Soar requires that a motor command be represented in Soar's belief language, which allows the agent to reason about and execute output commands using the same agent knowledge.

Systems that use completable plans may include conditional outputs (possibly in addition to other conditional actions). Soar conditionally decodes actions using the same computa-

tional processes that it uses for justified belief maintenance. The instantiated completion of an action is analogous to the automated elaboration of beliefs. Each framework supports methods for executing completable plans; some depending on plan language choices. Soar specifies what the plan language has to be, and therefore also specifies how plan completion occurs.

Processing Requirements

Unsurprisingly, each data element that appears in an agent also requires associated processing that the agent framework uses to activate (or select) and deactivate (or retract) that type of representation. To remain consistent with our goal of unifying the discussion across frameworks, we have examined each representational element in terms of how the framework manages *commitment to* and *reconsideration of* an associated data structure. Generalizing the notion of commitment and reconsideration across representational elements allows us to adopt a similar abstract-level view of processing for each framework, but focus on the aspects of processing in which each framework differs.

Justified Belief Maintenance

Justified beliefs receive no commitment from an agent beyond logical entailment. A set of justified beliefs must always be logically consistent with the elements from which the beliefs are deduced, in so far as the long-term logical rules that produce the beliefs are logically sound. In the BDI framework, this is where rational logic plays a key role. Soar realizes the encoding process with a reason maintenance system that automatically computes entailments from ground perceptions, assumptions, and previously computed entailments. Because reason maintenance is essentially a computational implementation of logic, it would make sense for BDI agents to use a similar implementation. As mentioned previously, augmented FSMs and GOMS may use variables to maintain and store justified beliefs, but the generic frameworks do not specify any particular approach or algorithm.

Assumption Maintenance

In contrast to justified beliefs, assumptions receive a very high level of commitment from agents. An assumption essentially remains in the belief set until the agent explicitly decides to remove it (or, in hierarchical representations, until the agent achieves or gives up the goals associated with the assumption). In BDI terms, this requires explicit intentions both to

create and to remove the assumption. Because it would be dangerous to create assumptions without some consideration, Soar demands that assumptions only be created as the result of deliberate intentions (whereas justified beliefs can be created by a more automatic process).

As this analysis demonstrates, the choice between justified beliefs and assumptions essentially boils down to the type of commitment or reconsideration necessary to activate or deactivate the beliefs. For rational agents, it may make sense to use justified beliefs as much as possible, in order to maintain a logically consistent belief set. This implies the use of a reason maintenance system, as Soar includes. Although BDI researchers have taken the question of commitment very seriously, they have mostly done so in terms of committing to (what they call) intentions and not to beliefs. The most popular BDI implementations do not include reason maintenance systems, even though logic and soundness are key parts of the BDI framework. GOMS and FSMs do not give the same prominent role to logic, and their implementations also do not generally include reason maintenance. In implementations that do not include reason maintenance, *all* beliefs are implemented as assumptions, and it is up to the agent developer to implement belief maintenance in the agent's knowledge base (for example, as part of each plan in a JAM agent).

Desire Maintenance

If an agent includes explicit representations of desires, there also needs to be a mechanism for maintaining desires. BDI again accomplishes this through a logical process. Presumably, modifications to support desires in the other frameworks could also mirror the processes that support belief maintenance. Soar includes a preference mechanism that allows the explicit proposal of actions that may not get selected, depending on the current context. One potential use for the preference mechanism would be to represent BDI-like desires, but that is not necessarily how the mechanism is used in practice. Similarly, desires could be represented and processed in GOMS or FSMs, but they would have to be maintained using specific knowledge encoded in the framework's action language.

Active Goal Maintenance

Active goals also require a mechanism of selection. In many frameworks, goal maintenance is similar to belief maintenance or arises as a side effect of the selection and execution of plans. For example, in GOMS goal activation occurs

by invoking the plan associated with a particular goal. Although this is a deliberative process, it is not divided into separate steps as in other frameworks. BDI and Soar include explicit processes for deliberate goal activation, distinct from the step of selecting a plan. FSMs, because they do not have explicit goals, also generally only have implicitly activated goals. An FSM goal may be considered active when the machine that implements that goal is executing. Therefore, we might say that an FSM activates a new goal by switching execution to a new plan (state machine).

Plan Maintenance

Tied closely to goal maintenance are questions of when an agent should select or abandon a current plan in the context of its current goals. Plan selection in most frameworks is fairly simple, with plans being indexed directly by goals. Under this model, activating a goal leads more or less directly to activating the associated plan. GOMS includes selection rules that allow the agent to consider different possible plans for a particular goal. As we have mentioned, Soar agents do not generally use explicit plan schemas, so it makes less sense to speak of plan selection for Soar. Rather, a "plan" in Soar is an emergent sequence of action selections.

For most of these frameworks the question of plan reconsideration is more interesting than the initial commitment to a plan. The question is when an agent should decide to abandon (or suspend) a plan, possibly even when it has chosen to remain committed to the plan's current goal (Wooldridge 2000). None of the frameworks provide clear advice on when such reconsideration should occur, but this is appropriate because it is a decision that requires knowledge of a particular task. The frameworks that use explicit plans in general must consider whether to provide support for the ability to abandon a plan reactively. The frameworks that do not directly support or insist on explicit plans implicitly support easier plan switching, because there is no separate commitment to a plan in the first place. Thus, in frameworks that do not require explicit plan representations, an agent may commit to one action at a time rather than committing to a whole plan (it is sometimes useful in such situations to view each action as a very fine-grained plan, or each plan as a very complex completable action). The BDI framework lies between the two extremes. While BDI insists that a plan must exist and be selected explicitly, it also dictates that plans should execute one action at a time, allowing time for reconsideration after each step. Some variations of GOMS and FSMs provide

similar functions.

The issue of commitment and reconsideration associated with plans has many ramifications for the design and capabilities of intelligent agents. If an agent uses completable plans, it is possible to commit to an abstract plan while allowing adaptations of the plan during execution. Each framework supports methods for executing completable plans, some depending on designer choices of a plan language. Soar specifies what the plan language has to be, and therefore also specifies how plan completion occurs. Agents that do not commit to high-level plans do not have to provide mechanisms for switching plans midstream. But they do have to include mechanisms for remaining committed to a particular course of action when necessary (without the benefit of support from an explicit plan).

Perhaps the most complex form of plan interruption involves switching back and forth between two or more plans in support of multiple goals (perhaps also meaning that the system is switching activation between the goals). This is clearly a capability of humans, and we mark it as a requirement for knowledge-intensive agents in many complex domains. The ability to accomplish task switching depends on the commitment and reconsideration methods for plan selection and execution but adds the problem of recommitting to a suspended plan. Neither BDI nor FSMs explicitly specify how such switching might occur. A model builder would have to encode a number of explicit conditions for when the plan should be abandoned and then taken up again. Additionally, it is not clear how a BDI agent should represent two goals that are actively being pursued, but in a switched manner. Frameworks that do not define these types of commitment and reconsideration leave the choices up to individual agent designers.

Related to plan switching is reentrant execution. It is sometimes desirable for an agent to resume a plan from a suspended point, rather than beginning the plan anew. Similarly, it can be advantageous to commit to portions of plans opportunistically when it appears that part of a plan is suddenly appropriate to a set of goals. Implementations of BDI or FSMs do not appear to support initiating the execution of a plan from somewhere in the middle of it. As suggested earlier, a possible alternative would be not to provide support for monolithic plans, as in Soar, and essentially treat each plan action atomically. Under such a scheme, instead of having explicit plan representations, each action must have appropriate selection conditions based on the belief set and active goals.

Many Soar agents implement task switching, but it is particularly difficult for GOMS and FSMs, since those frameworks explicitly insist on a hierarchical task structure. Switching between tasks in different parts of a task hierarchy requires quite a bit of overhead in continuously constructing and replacing the active hierarchies. In addition, an agent that is switching between two plans for two different goals must make sure that each plan's completions are sensitive to the effects of the other plan. This requirement demands some sort of shared memory in a global store or blackboard. The belief sets in BDI, GOMS, and Soar serve this purpose and enable communication between switching plans. However, even FSMs that allow variables and hierarchical decomposition generally encapsulate the variables within each machine, making such task switching onerous.

Plan Execution

A final issue involves how each framework constrains execution of a selected plan. Plan execution might also be called *action maintenance* because it has to do with the commitment to and reconsideration of the individual actions that make up the plan. The main issue here has to do with whether the process of execution integrates into the overall decision mechanisms of the framework. GOMS and Soar both represent plan actions as operators, which serve the dual purpose of executing primitive actions and activating new goals. Thus, they integrate the basic processes of reasoning and commitment into each execution step of a plan. BDI and FSM systems generally provide a plan language that is a complete programming language, relatively disconnected from the basic processes provided by the framework.

Certainly a plan language should contain methods for updating beliefs. Some implementations also include language primitives for creating desires and activating goals. Other features, such as loops, conditionals, and possibly local variables, provide very powerful execution abilities, but leave them outside of the constraints of the framework. This again imposes a trade-off between ease of programming (where the BDI and FSM implementations generally win) and taking advantage of the uniformity of the framework's built-in processes (where GOMS and Soar generally have an advantage).

Conclusions

The research communities that use agent frameworks continue to explore the issues that limit and inform the development of highly

competent intelligent agents *within* their frameworks (for example, Harland and Winikoff [2001]; Jones and Laird [1997]). However, too little attention is being paid to understanding the commonalities and differences across frameworks. Achieving this understanding is exacerbated by the differences in terminology and assumptions across research communities. We have attempted to contribute to this larger discussion by reviewing the directly supported representations and processes in broadly differing agent frameworks and adopting a rational, neutral, and common set of terms to describe each of them.

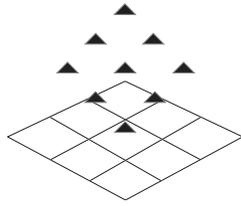
Each of these frameworks has been applied successfully to enough problems that it is not likely they are "missing" any representations and processes that are functionally necessary. However, from the point of view of creating deployed knowledge-intensive agents, the lack of explicit support from a framework imposes a burden on agent developers. Soar's (lack of) support for plans is a good example. The lack of an explicit plan representation lends flexibility in terms of plan execution (including interleaved execution with other plans). However, it also requires that a model builder create ad hoc solutions to plan commitment in the design of agent knowledge. Clearly this imposes a trade-off on the costs and benefits of using Soar's approach to plan representation. Each framework we have reviewed incorporates similar trade-offs with respect to various aspects of the framework's design.

Continuing to identify and develop representations and processes for agents is an important research activity. Increasingly, researchers are attending to processes necessary for social agents, including normatives, values, obligations, and teamwork. However, there are additional intraagent representations and processes that the frameworks discussed here do not directly support and that may be so widely necessary that they should be considered base-level representations. Examples include deliberate attention (Hill 1999), parallel active goals (Jones et al. 1994; Thangarajah, Padgham, and Harland 2002), and architectural support for managing resource limitations and conflicts (Meyer and Kieras 1997). Learning is also important for long-lived knowledge-intensive agents. The migration of knowledge into (and out of) long-term memory can also be studied in terms of representations, commitment, and reconsideration, resulting in a complex space of potential learning mechanisms (for example, along dimensions of automatic versus deliberate learning, or representations of procedural, declarative, and episodic memories).

This analysis lays the groundwork for extending and unifying the basic level representations and processes needed for knowledge-intensive intelligent agents. Perhaps as important, the analysis provides a potential theoretical framework and common set of terms to fuel future comparative and investigative work in the design of knowledge-intensive agent architectures.

References

- Albus, J. 2001. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. New York: John Wiley and Sons.
- Anderson, J. R.; and Lebiere, C. 1998. *Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum.
- Bratman, M. E. 1987. *Intentions, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press
- Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence* 4: 349–355.
- Card, S.; Moran, T.; and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum.
- Carmel, D.; and Markovitch, S. 1996. Learning Models of Intelligent Agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Ceranowicz, A., Nielsen, P.; and Koss, F. V. 2000. Behavioral Representation In JSAF. In *Proceedings of the Ninth Computer Generated Forces and Behavioral Representation Conference*. Orlando, FL: Institute for Simulation and Training, University of Central Florida.
- Chong, R. S.; and Wray, R. E. 2005. Constraints on Architectural Models: Elements of Act-R, Soar and Epic In Human Learning and Performance. In *Modeling Human Behavior with Integrated Cognitive Architectures: Comparison, Evaluation, and Validation*, ed. K. Gluck and R. Pew, 237–304. Mahweh, NJ: Lawrence Erlbaum Associates.
- d’Inverno, M.; Kinny, D.; Luck, M.; and Wooldridge, M. 1997. A Formal Specification of dMARS. In *Intelligent Agents IV Lecture Notes on Artificial Intelligence*, Volume 1365, ed. M. P. Singh, A. Rao, and M. J. Wooldridge, 155–176. Berlin: Springer Verlag.
- Forbus, K. D.; and deKleer, J. 1993. *Building Problem Solvers*. Cambridge, MA: The MIT Press.
- Freed, M. A.; and Remington, R. W. 2000. Making Human-Machine System Simulation a Practical Engineering Tool: An Apex Overview. Paper presented at the 2000 International Conference on Cognitive Modeling, Groningen, the Netherlands, 23–25 March.
- Gardenfors, P. 1988. *Knowledge In Flux*. Cambridge, MA: The MIT Press.
- Georgeff, M. P.; and Ingrand, F. F. 1990. Real-Time Reasoning: The Monitoring and Control of Spacecraft Systems. In *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications*. Piscataway, NJ: Institute of Electrical and Electronics Engineers.
- Georgeff, M. P.; and Lansky, A. L. 1987. Reactive Reasoning and Planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677–682. Menlo Park, CA: AAAI Press.
- Gervasio, M. T.; and DeJong, G. F. 1994. An Incremental Approach for Completable Planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 78–86. Menlo Park, CA: AAAI Press.
- Gray, W. D.; John, B. E.; and Atwood, M. E. 1993. Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Performance. *Human-Computer Interaction* 8(3): 237–309.
- Harland, J.; and Winikoff, M. 2001. Agents Via Mixed-Mode Computation In Linear Logic: A Proposal. Paper presented at the ICLP’01 Workshop on Computational Logic in Multi-Agent Systems. Paphos, Cyprus.
- Hill, R. 1999. Modeling Perceptual Attention In Virtual Humans. Paper presented at the Eighth Conference on Computer Generated Forces and Behavior Representation. Orlando, FL.
- Hopcroft, J. E.; and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.
- Howden, N.; Rönquist, R.; Hodgson, A.; and Lucas, A. 2001. Jack: Summary of an Agent Infrastructure. Paper presented at the Workshop on Infrastructure for Agents, MAS, and Scalable MAS at the Fifth International Conference on Autonomous Agents. Montreal, Canada.
- Huber, M. J. 1999. Jam: A BDI-Theoretic Mobile Agent Architecture. In *Proceedings of the Third International Conference on Autonomous Agents*, 236–243. New York: Association for Computing Machinery.
- John, B. E.; Vera, A. H.; and Newell, A. 1994. Toward Real-Time GOMS: A Model of Expert Behavior in a Highly Interactive Task. *Behavior and Information Technology* 13(4): 255–267.
- Jones, R. M.; and Laird, J. E. 1997. Constraints on the Design of a High-Level Model of Cognition. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*. Wheat Ridge, CO: Cognitive Science Society.
- Jones, R. M.; Laird, J. E.; Nielsen, P. E. 1999. Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine* 20(1): 27–42.
- Jones, R. M.; Laird, J. E.; Tambe, M.; and Rosenbloom, P. S. 1994. Generating Behavior in Response to Interacting Goals. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavior Representation*, 325–332. Orlando, FL: Institute for Simulation and Training, University of Central Florida.
- Kieras, D. E. 1997. A Guide to GOMS Model Usability Evaluation Using Ngomsl. In *Handbook of Human-Computer Interaction*, ed. M. Helander, T. Landauer, and P. Prabhu, 733–766. Amsterdam: North-Holland.
- Kieras, D. E.; Wood, S. D.; Abotel, K.; and Hornof, A. 1995. GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. In *Proceedings of the ACM Symposium on User Interface Software and Technology*. New York: Association for Computing Machinery.
- Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An Architecture for General Intelligence. *Artifi-*



AAAI 2007 Spring Symposium Series

The 2007 Spring Symposium Series will be held March 26 through March 31, 2007 at Stanford University. The Call for Participation will be available in July on the AAAI web site (www.aaai.org/Symposia/Spring/sss07.php).

Submissions will be due to the organizers on October 6, 2006.

For more information, please contact Symposium Chair, Alan Schultz, at schultz@aic.nrl.navy.mil or AAAI at sss07@aaai.org.

cial Intelligence 33(1): 1–64.

Lallement, Y.; and John, B. E. 1998. Cognitive Architecture and Modeling Idiom: A Model of the Wickens's Task. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*. Wheat Ridge, CO: Cognitive Science Society.

Langley, P.; Choi, D.; and Shapiro, D. 2004. A Cognitive Architecture for Physical Agents. Technical Report. Institute for the Study of Learning and Expertise, Palo Alto, CA.

Meyer, D.; and Kieras, D. 1997. Epic: A Computational Theory of Executive Cognitive Processes and Multiple-Task Performance: Part 1. *Psychological Review* 104: 3–65.

Newell, A. 1980. Physical Symbol Systems. *Cognitive Science* 4: 135–183.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18(1): 87–127.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Payne, T. R.; Singh, R.; and Sycara, K. 2002. Calendar Agents on the Semantic Web. *IEEE Intelligent Systems* 17(3): 84–86.

Peck, V. A.; and John, B. E. 1992. Browser-Soar: A Computational Model of a Highly Interactive Task. In *Proceedings, SIGChi Conference on Human Factors in Computing Systems*, ed. P. Bauersfeld, J. Bennett, and G. Lynch, 165–172. New York: ACM Press.

Rao, A.; and Georgeff, M. 1995. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multiagent Systems*. San Francisco. Menlo Park, CA: AAAI Press.

Russell S.; and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall. 1995

Schutt, M. C.; and Wooldridge, M. 2001. Principles of Intention Reconsideration. *Proceedings of the Fifth International Conference on Autonomous Agents*, 209–216. New York: ACM Press.

Thangarajah, J.; Padgham, L.; and Harland, J. 2002. Representation and Reasoning for Goals in BDI Agents. In *Proceedings of the Twenty-Fifth Australasian Conference on Computer Science — Volume 4*, Melbourne. Darlinghurst, Australia: Australian Computer Society.

Veloso, M. M.; Pollack, M. E.; and Cox, M. T. 1998. Rationale-Based Monitoring for Planning in Dynamic Environments. In *Proceedings of the Fourth International Conference on AI Planning Systems*. Menlo Park, CA: AAAI Press.

Wooldridge, M. 2000. *Reasoning about Rational Agents*. Cambridge, MA: The MIT Press.

Wray, R. E.; and Jones, R. M. 2005. An Introduction to Soar as an Agent Architecture. In *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, ed. R. Sun. Cambridge, UK: Cambridge University Press.

Wray, R. E.; and Laird, J. E. 2003. An Architectural Approach to Consistency In Hierarchical Execution. *Journal of Artificial Intelligence Research* 19: 355–398.



Randolph M. Jones, Ph.D., is a senior scientist at Soar Technology, Inc., and an assistant professor of computer science at Colby College. He has worked with a variety of agent architectures and models in both scientific research and applied intelligent systems. He has

more than 20 years of research and development experience with cognitive and agent architectures, intelligent agents, machine and human learning, graphical user interfaces, cognitive modeling, and a variety of related areas. Jones previously held research positions at the University of Michigan, the University of Pittsburgh, and Carnegie Mellon University. He earned a B.S. (1984) in mathematics and computer science at UCLA, and M.S. (1987) and Ph.D. (1989) degrees from the Department of Information and Computer Science at the University of California, Irvine. Contact rjones@soartech.com.



Robert E. Wray is chief scientist at Soar Technology. He received a Ph.D. in computer science and engineering from the University of Michigan. His doctoral research focused on maintaining logical consistency in agent reasoning systems, and his innovations were

incorporated in the Soar architecture. At Soar Technology, he leads or has led R&D projects for the U.S. Air Force, Army, and Navy and the Defense Advanced Research Projects Agency. Wray's research encompasses many areas of artificial intelligence including agent-based systems and agent architectures, machine learning, cognitive science, and knowledge