

Toward an Abstract Machine Architecture for Intelligence

Randolph M. Jones and Robert E. Wray

Soar Technology, Inc.
3600 Green Court, Suite 600
Ann Arbor, MI 48105

rjones@soartech.com, wray@soartech.com

Abstract

Our research defines a uniform framework for analyzing a representative sampling of existing architectures and frameworks for knowledge-intensive intelligent agents. We use this framework to build abstract definitions of representational and functional components that are common across architectures. Defining these abstract components essentially allows us to describe a target abstract machine architecture for knowledge-intensive intelligent systems. This abstract layer should provide a useful tool for comparing and evaluating architectures, as well as for building higher-level languages and tools that reduce the expense of developing knowledge-intensive agents.

Architectures for Intelligent Systems

Our current work follows a strong analogy between intelligent agent architectures and the system engineering of “standard” architectures (e.g., von Neumann machines) for computation. The basic insight is that cognitive architectures today provide various instantiations of “virtual machines” for intelligent systems. Like standard computing architectures, they provide fundamental computational pieces that are generally fine-grained, comprehensive, and (in most cases) efficient. Additionally, cognitive architectures, like their von Neumann counterparts, generally provide a low-level programming language that allows direct access to architectural components. These programming languages are akin to assembly languages for other computing platforms, although the computational models they support are generally quite distinct from their more standard and procedural counterparts, which we outline in this paper.

These low-level programming languages have sufficed for intelligent agent research programs, as well as deployed models of “light” agents that do not contain copious amount of code. However, there has been increasing interest in building very large, knowledge-intensive intelligent agent systems (eg., Jones et al. 1999). The difficulty in attempting to build and maintain such systems in “assembly language” should be painfully obvious. If we

are to continue building such systems with even a modicum of frugality, efficiency, and robustness, we must engineer higher-level languages for intelligent agent systems, as our colleagues in software engineering have done for non-agent programming.

However, in the design of such high-level languages, we should also pay attention to the lessons of software engineering. Although the high-level languages should be powerful and programmer-friendly, they also need to map well to the underlying architecture. Standard computing architectures share a number of similarities, and the development of the Java framework has demonstrated the utility of abstracting across the implementation differences in various computers to define a *virtual machine*, or an *abstract machine layer* that provides a uniform interface to various particular computing platforms.

This paper outlines initial work on a similar abstract machine layer for intelligent agent architectures, and ultimately their high-level languages and development tools. We have approached this task by first analyzing a sampling of agent architectures to identify the particular architectural components that define them and to find abstractions across the various particular implementations of functional components. We have also undertaken this analysis by developing a uniform view of how intelligent agent architectures bring various types of knowledge to bear on decision making and behavior. Our goal is to formalize a set of abstract components that can be used to define an abstract machine layer for intelligent systems.

Focus on cognitive architectures

For the purposes of this research, we are interested in the development of *knowledge-intensive agents*. We use this term primarily in order to distinguish this type of agent from the popular notion of *light* agents (which are often developed and used in the multi-agent systems community). One defining (and desirable) characteristic of light agents is that each individual agent is fairly simple, having very specific, narrow goals and constrained interactive abilities. Examples of light agents include brokers and matchmakers for E-commerce applications. Light agents solve complex problems through the emergent interactions of large groups of the simple agents (the

formation of a market via the interaction of a group of light agents).

However, in many cases individual computer programs must address complex reasoning tasks on their own, requiring agents that encode a lot of knowledge about a problem, and have a broad range of capabilities for solving them. Problems most appropriate for knowledge-intensive agents are characterized by a complex variety of interacting goals and a comparatively large number of modes of interaction with the environment. In response to this level of complexity, knowledge-intensive agents generally require explicit representations of beliefs, goals, and intentions, but these must further be structured into sophisticated relational (and often hierarchical), structured representations of situational awareness. In addition, such agents generally must be capable of mixed levels of commitment and interaction, generating appropriate combinations of purposeful and reactive behaviors.

The performance and reasoning requirements of these problems drive the types of primitive computational elements that are best suited for the development of such agents. Cognitive architectures reflect explicit attempts to support the required primitive computational elements for knowledge-intensive agents. For example, human behavior models that perform in complex and dynamic environments require autonomy and flexibility in execution. Existing cognitive architectures directly provide and support non-linear control constructs that aim to address such capabilities. Such control constructs include productions (Newell 1973) and other efficient, relational pattern matchers; rapid context switching for interruptibility and pursuit of simultaneous goals; and varying amounts of parallelism in pattern matching, reasoning, and behavior. Other important computational elements supported by cognitive architectures include structured relational representations of situational awareness; automated belief maintenance; least-commitment planning and execution (Weld 1994), and architectural support for learning.

Least commitment is a fundamental requirement for autonomous, flexible, adaptable behavior, but results in sharp contrasts with traditional software engineering. Least commitment refers to the postponement of a decision until the decision actually needs to be acted upon. Least commitment allows agents to make context-sensitive decisions about behavior and resource allocations, and also to be flexible about adapting those decisions in the face of a changing environment or assumptions. Weld contrasts least commitment mechanisms, in which control decisions are made at every decision opportunity, with traditional control logic, in which control decisions are hard-coded when the program is designed and compiled. These differences are illustrated in Figure 1.

For an agent to be autonomous and robust, it must be able to make decisions based on the context in which it finds itself. Therefore, unlike traditional software engineering, the role of the knowledge engineer is not to program “good” decisions, but to give the agent the ability to make decisions in its context. Design-time and compile-time decisions that limit run-time flexibility must be

minimized. Least-commitment requires conflict-resolution solutions that are mediated in the context, rather than a fixed procedure, such as the rule selection techniques used in typical rule-based systems (Forgy 1981). Less obviously, least commitment also requires *weak encapsulation*. Strong typing and encapsulation result from design-time decisions about the appropriate objects for an agent to consider when making decisions. In the long run, the agent must have the flexibility to decide for itself which of its knowledge is applicable to a local decision.

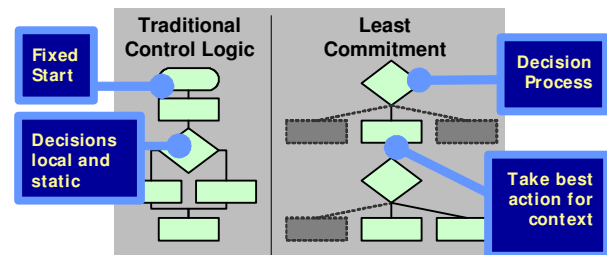


Figure 1: Least commitment in comparison to traditional control logic.

Cognitive architectures also generally provide explicit mechanisms for relating parallel processing (for example, at the level of memory retrieval, pattern matching, or analysis of courses of action) to serial processing (where behavior systems must ultimately generate a serial set of commitments to action). In essence, cognitive architectures define processes to support knowledge-driven commitments to courses of action, mixed with appropriate reactive context switching when necessary.

Knowledge in cognitive architectures normally is encoded *associatively*, as opposed to *sequentially* or *functionally*, as is standard practice in procedural computing architectures. Each cognitive architecture includes some mechanism for associative retrieval of potential courses of action, and then a conflict resolution mechanism for choosing between the candidates. We argue (and research into cognitive architectures seems to confirm) that associatively represented knowledge is a fundamental key to capturing the types of mixed-initiative commitment to action that are expected of artifacts with human-like intelligence.

A final reason to focus on cognitive architectures is that they generally attempt to provide at least some account of all aspects of intelligent behavior, and provide explicit structures and processes for modeling them. This breadth particularly includes learning and long-term adaptation to new environments. Learning will be a key part of future development of sophisticated human behavior models. Much additional research is needed before learning is used in robustly engineered, knowledge-intensive agents. However, learning is likely a requirement for autonomous, flexible systems, and successful efforts to design abstract frameworks for intelligent agents must address the challenges of learning early in design.

Building high-level abstractions

We are investigating existing cognitive architectures in order to identify properties, representations, and mechanisms that they share and to create abstractions of these shared computational features. The abstraction process will help rationalize and unify work by researchers active in intelligent agents, cognitive architectures, and human behavior representation. By focusing on general categories of functions and representations within the scope of knowledge intensive, situated agents, we can explicitly identify and study much of what has been learned about computational intelligence.

Abstraction will also enable us to specify abstract architectural components that are independent of the details of any particular cognitive architecture. This in turn should support improved levels of abstraction in the languages and tools used to build intelligent agent models. Currently, model implementation must be performed by a knowledge engineer who is intimately familiar with the finest details of the implementation architecture, similar to a software engineer using assembly language to interface directly to a hardware architecture. Additionally, once a model has been implemented, it will necessarily include many design features that closely tie it to the implementation architecture, which makes transfer of the model to other applications, architectures, or environments more difficult.

This problem is familiar; it has been the subject of research in software engineering. One software-engineering solution to this problem has been to define a virtual machine layer that allows computer languages (such as Java) to support high-level program design that functions across low-level architectures. We hope to apply this notion of a virtual machine to cognitive architectures, so that programs developed in a high-level cognitive architecture language can be applied across architectures in the development and application of knowledge-intensive intelligent systems.

Thus far, we have identified and enumerated a number of functional patterns and development patterns across existing cognitive architectures, as well as some additional intelligent system frameworks. Our most thorough analysis focuses on patterns within the Soar (Laird and Rosenbloom 1995; Laird, Newell, and Rosenbloom 1987; Newell 1990) and ACT-R (Anderson and Lebiere 1998) architectures, but our work has also been informed by patterns in other frameworks for intelligent systems, such as belief-desire-intention (BDI) (Bratman 1987; Wooldridge 2000) planning systems (Erol, Hendler, and Nau 1994; Weld, Anderson, and Smith 1998), and the GOMS psychological modeling paradigm (Card, Moran, and Newell 1983).

Commitment and reconsideration

To perform our analysis, we have developed a general framework for analyzing the processes of cognitive architectures. We call the framework *CCRU*, for

Consideration, Commitment, Reconsideration, and Unconsideration. Our framework generalizes the notions of commitment (Wooldridge 2000) and reconsideration (Schutt and Wooldridge 2001) in intelligent agents. Each behavioral function within a model can be viewed as the commitment to or reconsideration of a particular type of knowledge structure. This guiding principle allows us to create abstract processes for each type of knowledge structure, and then further to categorize various implementations of processing by formally defining the commitment and reconsideration procedures provided by each cognitive architecture. Analysis of commitment and reconsideration procedures is important for the least-commitment paradigm that identifies knowledge-intensive agents. Functional or sequential formalisms violate the notion of least commitment because they represent decisions made outside of the performance context. Instead we uniformly formalize the commitment and reconsideration strategies for each representational structure that makes up an agent's knowledge base. In addition, these dual levels of analysis provide exactly the information we would need to design an abstract machine layer for intelligent agent architectures, as well as higher-level agent languages (Crossman et al. 2004) and various *knowledge compilers* that could translate models into the virtual machine (based around the various implementations of commitment and reconsideration).

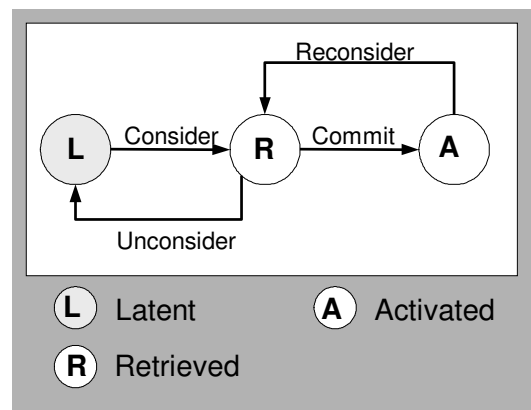


Figure 2: Unifying view of memory operations via consider, commit, reconsider, unconsider, and store.

CCRU defines four basic processes that address the ways knowledge structures are handled within an intelligent agent model. Most of our work to date has focused on knowledge representations that support decision-making in an agent's dynamic memory. However, we believe this analysis will extend to various types of long-term knowledge, as well. The four basic processes correspond to transformation between three knowledge-structure states, as shown in Figure 2. Our labels for these three knowledge states are *latent*, *retrieved*, and *activated*. The following four processes govern the various transitions that can change a single knowledge structure's state.

Consideration is a retrieval process that instantiates a specific knowledge structure from an outside stimulus or

long-term memory. Here “knowledge structure” is defined as any mental structure that an agent can reason about, such as representations of real world objects, goals, and transforms. For example, after a factual knowledge structure has been considered (and becomes retrieved), it is available for reasoning, but is not yet part of the agent’s beliefs (i.e., it is not activated). The process of consideration moves a particular knowledge structure from the **Latent** state to the **Retrieved** state. Consideration is a transformation process, but implies an underlying creation process for constructing memory representations. Architectures implement the details of the creation process in different ways, but share the transformational concept of retrieving a knowledge structure for further reasoning. Examples of consideration include an agent considering a new decision-making goal, or considering whether to believe a new interpretation of the environment.

Commitment is the process of committing to a particular knowledge structure. After a knowledge structure is committed, we say it is *activated*. Once activated, it becomes a part of the current decision-making context and can be used as a basis for making further decisions. This process moves a particular knowledge structure from the **Retrieved** state to the **Activated** state. Commitment does not change the contents of memory, but rather the state of what is in memory, as driven by a deliberate decision-making process. For example, after retrieving a number of potential goals to pursue, an agent may commit to a particular goal after deliberately determining that it is the goal most likely to direct productive action in a particular situation. In general, an agent will not reactively commit to a goal without reasoning over it first. For example, an agent generally should not commit to a goal based purely on an external stimulus, as that can lead to a lack of coherence in behavior.

Reconsideration is the process of removing a knowledge structure from the current decision-making context. The agent can still reason about a deactivated knowledge structure, because the structure is still retrieved, but it cannot be used as a precondition for further decision making. Reconsideration moves a knowledge structure from the **Activated** state back to the **Retrieved** state. Examples of reconsideration include an agent deactivating a goal when it discovers the goal is currently unachievable, or discounting a belief in a particular interpretation of the environment after encountering some disconfirming evidence. Reconsideration is also important for coherence in behavior. When an agent is interrupted, it may deactivate some previously active goal. However, once the interruption is complete, the agent may want to commit again to the previous goal. Because that goal remains retrieved (it has not been unconsidered), the context associated with the previous goal remains available and does not require repeated instantiation or elaboration.

In our initial formulation (as presented here), we do not make a distinction between newly considered objects and those that had been previously activated, but have been “demoted” to the considered state. However, in

implementing an initial compiler for a language that expresses CCRU, we have found that distinguishing between newly considered and reconsidered objects is sometimes necessary. This may lead, in the future, to some elaboration of the CCRU states presented in Figure 2.

Unconsideration is the process of removing a knowledge structure entirely from the portion of memory that is directly accessible to decision making. Unconsideration moves a particular knowledge structure from the **Retrieved** state to the **Latent** state. Unconsideration, similar to consideration, is a transformation process that implies an underlying memory management process for removing memory representations from the current context. Examples of unconsideration include an agent removing an achieved or unachievable goal, or deleting a belief that is not consistent with other, more recently activated beliefs.

Review of Knowledge-Intensive Agent Frameworks

The CCRU framework provides a general and uniform capability to analyze the various knowledge components implemented by a particular cognitive architecture or abstract machine layer. In the context of this framework, we have analyzed a number of knowledge components in some existing architectures and agent frameworks. The point of this exercise is to construct empirically a set of abstract representations and processes that appear to be commonly found in systems that exploit (or at least support) knowledge-intensive reasoning. By collecting instances of various types of components, and casting them into the uniform CCRU framework, we will begin to define the design of an abstract machine layer consisting of a comprehensive set of integrated components.

Below we review three mature frameworks for intelligent agents that represent three different theoretical traditions (philosophical and logical, functional, and psychological). We introduce the primary representational constructs and processes directly supported by each and map them to CCRU.

Beliefs-Desires-Intentions (BDI)

The Beliefs-Desires-Intentions (BDI) framework is a logic-based methodology for building competent agents (Georgeff and Lansky 1987; Rao and Georgeff 1995; Schutt and Wooldridge 2001). 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 proven logically. In BDI, actions arise from internal constructs called intentions. An intelligent agent cannot make decisions about intentions until it has at least some representation of its environment and situation. 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,

called intentions, to pursue. A BDI agent's actions must be logically consistent with its combination of beliefs and goals. This property is not generally true of all frameworks (particularly when they are used for psychological models). We will mention examples from implemented architectures, but our primary consideration of BDI is based on the general 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 formalizes many details of high-level human reasoning and interaction. GOMS is particularly interesting because knowledge-intensive agents are often used to simulate human behavior. In addition, because implemented cognitive architectures in general are getting faster (John, Vera, and Newell 1994), they will increasingly compete with AI architectures as platforms for agent development.

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) 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. Like BDI, GOMS is a high-level framework, realized in a number of individual implementations.

Soar

Soar has roots in cognitive psychology and computer science, but it is primarily a functional approach to encoding intelligent behavior {Laird, 1987 #1}. A strong focus of Soar research is to identify 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 agent 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 1980a) and the physical symbol systems hypothesis (Newell 1980b). Problem spaces modularize long-term knowledge so that it can be brought to bear in a goal-directed series of discrete steps. 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 problem space hypothesis relies on an assumption of rationality, similar to BDI. In many Soar agent systems, problem spaces implement hierarchical task representations, where each portion of the hierarchy represents a different problem space, although problem spaces can also support other types of context switching. The physical symbol systems

hypothesis led to Soar's commitment to uniform representations of knowledge and beliefs.

While Soar imposes strong constraints on fundamental aspects of intelligence, it does not impose functionally inspired high-level constraints (in the spirit of BDI's use of logic, or GOMS' use of hierarchical goal decomposition). Thus, Soar is a lower-level framework for reasoning than BDI and GOMS. Either BDI principles or GOMS principles can be followed when using Soar as the implementation architecture.

ANALYSIS OF HEAVY 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, from the point of view of an implemented agent system developed within any of these frameworks, the designer has to make many more decisions about agent construction than provided by each framework's core principles, essentially working at an "assembly language" level instead of a "high language" level.

Consider this group of frameworks together. None give first-class attention to the union of necessary representations and processes. In any agent of sufficient complexity and intelligence, each of these high-level features must be addressed.

From a software engineering perspective, an abstract machine for intelligent agents should provide a set of general, reusable elements that transfer from one agent application to another. However, to maximize reuse, any representational element or process that is useful in most agent applications should be addressed in the framework. This philosophy is guiding our development of the definition of an abstract machine architecture. Certainly, the union of BDI, GOMS, and Soar will not provide *all* possible first-level design principles; however, they provide a significant starting point, and define a methodology we will use to incorporate elements from additional agent frameworks.

Table 1 lists the union of the base-level representations from BDI, GOMS, and Soar, unified by functional roles. The representations are ordered to suggest the basic information flow from an external world into agent reasoning and then back out. The "representation language" 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 use the CCRU formalism 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 and "reconsider" to indicate the process of determining whether a commitment should be withdrawn.

Table 1. Agent framework comparisons. Black items are specific solutions provided by the framework. Grey items are general support provided by the framework. No entry means the framework does not explicitly address the element.

Base-level Representation	Representation Language	Commitment	Reconsideration
Inputs	<i>BDI</i> Input language <i>GOMS</i> Input language <i>Soar</i> Working memory		
Justified Beliefs	<i>BDI</i> Beliefs <i>GOMS</i> Working memory <i>Soar</i> Working memory	Logical inference Match-based assertion Matched-based assertion	Belief revision Reason maintenance
Assumptions	<i>BDI</i> Beliefs <i>GOMS</i> Working memory <i>Soar</i> Working memory	Plan language Operators Deliberation/Operators	Plan language Operators Operators
Desires	<i>BDI</i> Desires <i>GOMS</i> Proposed operators <i>Soar</i> operators	Logic Preferences	Logic Preferences
Active Goals	<i>BDI</i> Intentions <i>GOMS</i> Goals <i>Soar</i> Beliefs & Impasses	Deliberation Operators Deliberation	Soundness/Decision Theory Reason maintenance
Plans	<i>BDI</i> Plans <i>GOMS</i> Methods <i>Soar</i>	Plan selection Selection	Soundness Interleaving
Actions	<i>BDI</i> Plan language <i>GOMS</i> Operators Primitive <i>Soar</i> Operators	Atomic actions Operators Deliberation	 Reason maintenance
Outputs	<i>BDI</i> <i>GOMS</i> Plan language <i>Soar</i> Working memory	Plan language Conditional operators	

CCRU allows us to classify *all* of the processes that operate on the base-level representational components and to see relationships between different approaches. For example, all the frameworks use an explicit deliberation process to commit to goals, although the reconsideration process for goals is quite different. BDI uses decision theory, Soar, reason maintenance, and GOMS does not specify a reconsideration process, suggesting that goals must be

reconsidered within the context of the basic reasoning machinery. These points of similarity and difference illustrate how an abstract machine architecture could be developed, and also some of the challenges in implementing agent programs developed in the virtual machine language on the architectural platforms.

Conclusion

We have described our initial efforts to define a set of abstract functional components that describe the representations and processes common across a variety of architectures and frameworks for knowledge-intensive intelligent agents. We have collected and analyzed these components by defining a uniform framework, which we call CCRU, for characterizing how intelligent agent systems bring each of their various types of knowledge to bear on decision-making processes.

By defining these abstract components, we hope to provide a middle abstract machine layer between particular cognitive architectures and agent frameworks (which already exist) and new high-level languages and tools for the development of knowledge-intensive agents (which are in demand). Such an abstract machine definition should guide the construction of new languages, allowing them to provide generality, power, and expressiveness, while maintaining a connection to the underlying functional components required by knowledge-intensive agents.

This research can also be of benefit to the study of cognitive architectures in their own right. Even if our abstract machine definition is not used for the development of high-level languages, it makes clear that there is well defined level of abstraction from which intelligent agent architectures can be viewed to be quite similar to each other. This level of abstraction can provide use with a common reference point and language for comparing architectures, evaluating them, and developing new architectures.

References

- 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.
- Card, S., Moran, T., and Newell, A. 1983. *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum.
- Crossman, J., Wray, R. E., Jones, R. M., and Lebiere, C. 2004. A high level symbolic representation for behavior modeling. In K. Gluck (Ed.) *Proceedings of the 2004 Conference on Behavioral Representation in Modeling and Simulation*. Arlington, VA.
- Erol, K., J. Hendler, N., et al. (1994). *HTN planning: Complexity and expressivity*. 12th National Conference on Artificial Intelligence.
- Forgy, C. L. 1981. *OPS4 user's manual*. Technical report no. CMU-CS-81-135. Carnegie Mellon University Computer Science Department.
- Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677-682. Menlo Park, CA: AAAI Press.
- 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., Laird, J. E., Nielsen, P. E. et al. 1999. Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine*. 20(1): 27-42.
- Laird, J. E., Newell, A., and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33(1): 1-64.
- Laird, J. E., and Rosenbloom, P. S. 1995. The evolution of the Soar cognitive architecture. In T. Mitchell (Ed.) *Mind Matters*. Hillsdale, NJ: Lawrence Erlbaum.
- Newell, A. (1973). *Production Systems: Models of Control Structures*. Visual Information Processing. W. Chase. New York, Academic Press.
- Newell, A. 1980a. Reasoning, problem solving, and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.) *Attention and Performance VIII*. Hillsdale, NJ: Lawrence Erlbaum.
- Newell, A. 1980b. Physical symbol systems. *Cognitive Science* 4:135-183.
- Newell, A. 1990. *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
- Rao, A., and Georgeff, M. 1995, BDI agents: From theory to practice. *Proceedings of the First Intl. Conference on Multiagent Systems*. San Francisco.
- 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
- Weld, D. (1994). An Introduction to Least Commitment Planning. *AI Magazine* 15(4): 27-61.
- Weld, D. S., C. R. Anderson, et al. 1998. *Extending Graphplan to Handle Uncertainty & Sensing Actions*. Fifteenth National Conference on Artificial Intelligence, Madison, Wisconsin.
- Wooldridge, M. (2000). *Reasoning about Rational Agents*. Cambridge, MA: MIT Press..