AI Magazine Volume 27 Number 2 (2006) (© AAAI)



# Cognitive Architectures and General Intelligent Systems

*Pat Langley*

■ In this article, I claim that research on cognitive architectures is an important path to the development of general intelligent systems. I contrast this paradigm with other approaches to constructing such systems, and I review the theoretical commitments associated with a cognitive architecture. I illustrate these ideas using a particular architecture—ICARUS—by examining its claims about memories, about the representation and organization of knowledge, and about the performance and learning mechanisms that affect memory structures. I also consider the high-level programming language that embodies these commitments, drawing examples from the domain of in-city driving. In closing, I consider ICARUS's relation to other cognitive architectures and discuss some open issues that deserve increased attention.

## The Need for General Intelligent Systems

The original goal of artificial intelligence was the design and construction of computational artifacts that combined many cognitive abilities in an integrated system. These entities were intended to have the same intellectual capacity as humans and they were supposed to exhibit their intelligence in a general way across many different domains. I will refer to this research agenda as aimed at the creation of *general intelligent systems*.

Unfortunately, modern artificial intelligence has largely abandoned this objective, having instead divided into many distinct subfields that care little about generality, intelligence, or even systems. Subfields like computational linguistics, planning, and computer vision focus their attention on specific components that underlie intelligent behavior, but seldom show concern about how they might interact with each other. Subfields like knowledge representation and machine learning focus on idealized tasks like inheritance, classification, and reactive control that ignore the richness and complexity of human intelligence.

The fragmentation of artificial intelligence has taken energy away from efforts on general intelligent systems, but it has led to certain types of progress within each of its subfields. Despite this subdivision into distinct communities, the past decade has seen many applications of AI technology developed and fielded successfully. Yet these systems have a "niche" flavor that differs markedly from those originally envisioned by the field's early researchers. More broadly based applications, such as human-level tutoring systems, flexible and instructable household robots, and believable characters for interactive entertainment, will require that we develop truly integrated intelligent systems rather than continuing to focus on isolated components.

As Newell (1973) argued, "You can't play

twenty questions with nature and win." At the time, he was critiquing the strategy of experimental cognitive psychologists, who studied isolated components of human cognition without considering their interaction. However, over the past decade, his statement has become an equally valid criticism of the fragmented nature of AI research. Newell proposed that we move beyond separate phenomena and capabilities to develop complete models of intelligent behavior. Moreover, he believed that we should demonstrate our systems' intelligence on the same range of domains and tasks as handled by humans, and that we should evaluate them in terms of generality and flexibility, rather than success on a single domain. He also viewed artificial intelligence and cognitive psychology as close allies with distinct yet related goals that could benefit greatly from working together. This proposal was linked closely to his notion of a *cognitive architecture,* an idea that I can best explain by contrasting it with alternative frameworks.

## Three Architectural Paradigms

Artificial intelligence has explored three main avenues to the creation of general intelligent systems. Perhaps the most widely known is the *multi-agent systems* framework (Sycara 1998), which has much in common with traditional approaches to software engineering. In this scheme, one develops distinct modules for different facets of an intelligent system, which then communicate directly with each other. The architecture specifies the inputs/outputs of each module and the protocols for communicating among them, but places no constraints on how each component operates. Indeed, the ability to replace one large-scale module with another equivalent one is viewed as an advantage of this approach, since it lets teams develop them separately and eases their integration.

One disadvantage of the multi-agent systems framework is the need for modules to communicate directly with one another. Another paradigm addresses this issue by having modules read and alter a shared memory of beliefs, goals, and other short-term structures. Such a *blackboard system* (Engelmore and Morgan 1989) retains the modularity of the first framework, but replaces direct communication among modules with an indirect scheme that relies on matching patterns against elements in the short-term memory. Thus, a blackboard architecture supports a different form of integration than the multiagent scheme, so that the former comes somewhat closer to theories of human cognition.

However, Newell's vision for research on integrated theories of intelligence included more than either of these frameworks provides. He believed that agent architectures should incorporate strong theoretical assumptions about the nature of the mind. An architectural design should change only gradually, as one determines that new structures and processes are required to support new functionality. Moreover, early design choices should constrain heavily those made later, producing far more interdependence among modules than assumed by either multiagent or blackboard systems. Newell (1990) claimed that architectural research is all about mutual constraints, and its aim should be a *unified* theory of intelligent behavior, not merely an integrated one.

The notion of a *cognitive architecture* revolves around this interdependent approach to agent design. Following Newell's lead, research on such architectures makes commitments about: (1) the short-term and long-term memories that store the agent's beliefs, goals, and knowledge; (2) the representation and organization of structures that are embedded in these memories; (3) the functional processes that operate on these structures, including both performance and learning mechanisms; and (4) a programming language that lets one construct knowledge-based systems that embody the architecture's assumptions. These commitments provide much stronger constraints on the construction of intelligent agents than do alternative frameworks, and they constitute a computational theory of intelligence that goes beyond providing a convenient programming paradigm.

In the next section, I will use one such cognitive architecture—ICARUS—to illustrate each of these commitments in turn. ICARUS is neither the oldest nor the most developed architecture; some frameworks, like ACT (Anderson 1993) and Soar (Laird, Newell, and Rosenbloom 1987), have undergone continual development for more than two decades. However, it will serve well enough to make the main points, and its differences from more traditional cognitive architectures will clarify the breadth and diversity of this approach to understanding the nature of intelligence.

In discussing ICARUS, I will draw examples from the domain of in-city driving, for which we have implemented a simulated environment that simplifies many aspects but remains rich and challenging (Choi et al. 2004). Objects in this environment include vehicles, for which the positions, orientations, and velocities change over time, as well as static objects like road segments, intersections, lane lines,

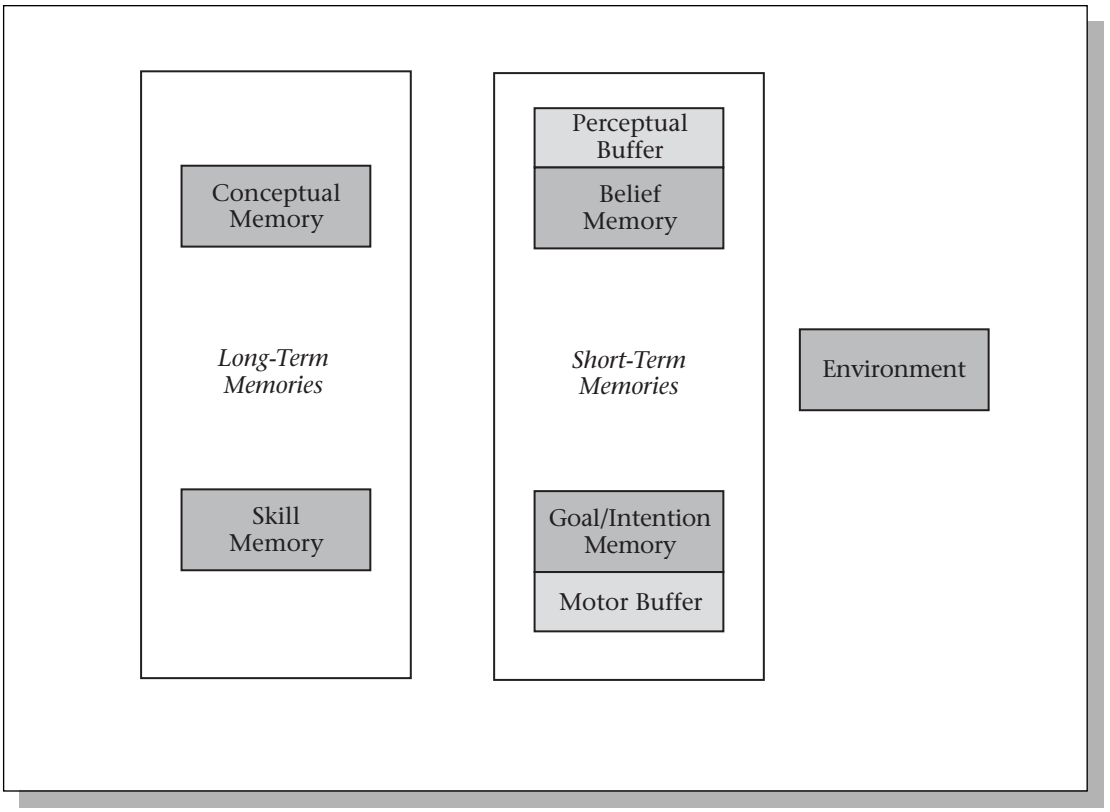


*Figure 1. Six Long-Term and Short-Term Memories of the ICARUS Architecture.*

sidewalks, and buildings. Each vehicle can alter its velocity and change its steering wheel angle by setting control variables, which interact with realistic laws to determine each vehicle's state. We have implemented ICARUS agents in other domains, but this is the most complex and will serve best to communicate my main points.

## The ICARUS Architecture

As noted above, ICARUS is a cognitive architecture in Newell's sense of that phrase. Like its predecessors, it makes strong commitments to memories, representations, and cognitive processes. Another common theme is that it incorporates key ideas from theories of human problem solving, reasoning, and skill acquisition. However, ICARUS is distinctive in its concern with physical agents that operate in an external environment, and the framework also differs from many previous theories by focusing on the organization, use, and acquisition of hierarchical structures. These concerns have led to different assumptions than those found in early architectures such as ACT and Soar.

Our research on ICARUS has been guided by five high-level principles about the nature of general intelligent systems: (1) cognition is grounded in perception and action; (2) concepts and skills are distinct cognitive structures; (3) long-term memory is organized in a hierarchical fashion; (4) skill and concept hierarchies are acquired in a cumulative manner; and (5) long-term and short-term structures have a strong correspondence. These ideas further distinguish ICARUS from most other cognitive architectures that have been developed within the Newell tradition. Again, I will not claim here that they make the framework superior to earlier ones, but I believe they do clarify the dimensions that define the space of candidate architectures.

### Memories and Representations

To reiterate, a cognitive architecture makes a commitment to the memories that store the content and that control its behavior. These must include one or more long-term memories that contain knowledge and procedures, along with one or more short-term memories that store the agent's beliefs and goals. The contents of long-term memories[1] change gradually or not at all, whereas short-term elements change rapidly in response to environmental conditions and the agent's agenda. Some architec-

tures also incorporate sensorimotor memories that hold information about perceptions and actions; these are updated rapidly as the system perceives new objects and executes its procedures.

As figure 1 depicts, ICARUS includes six distinct memories, two of each variety. Unlike traditional production-system architectures, which encode all long-term contents as condition-action rules, it has two separate memories, one for conceptual knowledge and another for skills or procedures. The framework has two analogous short-term memories, one for the agent's beliefs about the environment and another for its goals and associated intentions. Finally, ICARUS has a perceptual buffer that holds immediate perceptions of the environment and a motor buffer that contains skills and actions it intends for immediate execution.

ICARUS's focus on physical settings distinguishes it from traditional cognitive architectures, including early versions of Soar and ACT, although both frameworks have since been extended to interface with external environments. For example, Laird and Rosenbloom (1990) report a variant of Soar that controls a physical robot, whereas Byrne (2001) describes ACT-R/PM, which augments ACT-R with perceptual and motor buffers. However, both theories focused initially on central cognition and added other modules at a later date, whereas ICARUS began as an architecture for reactive execution and places greater emphasis on interaction with the physical world.

In addition to positing memories, a cognitive architecture makes theoretical claims about the representations used to store information in those memories. Thus, it commits to a particular syntax for encoding long-term and short-term structures. Most frameworks rely on formalisms similar to the predicate calculus that support expression of relational content. These build on AI's central assumption that intelligent behavior involves the manipulation of symbolic list structures (Newell and Simon 1976). An architecture also specifies the manner in which it organizes these structures in memory, along with the way those structures are connected across different memories.

For instance, ICARUS represents the contents of long-term conceptual memory as Boolean concepts that encode knowledge about classes of objects and relations among them. Each concept definition includes a head, which specifies its name and arguments, and a body, which includes a *:percepts* field that describes the types and attribute values of observed perceptual entities, a *:relations* field that states lower-level concepts it must match or must not match, and a *:tests* field that specifies Boolean tests it must satisfy.

Table 1 shows some concepts from the driving domain. For example, the nonprimitive concept *driving-well-in-segment* takes three arguments: ?*self, ?seg,* and *?lane.* The *:percepts* field indicates that the first refers to the agent itself, the second denotes a road segment, and the last refers to a lane line associated with the segment. Each structure must match against elements in the perceptual buffer, which I describe below. The *:relations* field states that the agent must hold five interconnected beliefs about these entities before it can infer an instance of the concept.

Another nonprimitive concept in the table, *in-rightmost-lane,* shows that the *:relations* field can also contain negated conditions. These match only when the agent does not hold any corresponding belief, with unbound variables like *?anylane* being treated as universally quantified. The table also illustrates two primitive concepts, *in-lane* and *in-segment,* that refer to perceived entities and Boolean tests on their attribute values, but that do not refer to any simpler concepts. ICARUS concepts are very similar to Horn clauses in the programming language Prolog (Clocksin and Mellish 1981), although the syntax differs somewhat.

In contrast, long-term skill memory encodes knowledge about ways to act and achieve goals. Each skill clause has a head, which specifies a concept the clause should achieve upon successful completion, and a body with a variety of fields. For primitive skill clauses, the body includes a *:start* field that describes the situation in which the agent can initiate the clause, a *:requires* field that must hold throughout the skill's execution (for use in durative skills), and an *:actions* field that indicates executable actions the clause should invoke. For example, Table 2 shows a primitive skill clause for *in-intersection-for-right-turn,* which is considered only when one start condition *(in-rightmost-lane)* holds at the outset and when three requirements hold throughout execution.

Nonprimitive skill clauses differ from primitive ones in that they have no *:actions* field, since they instead have a *:subgoals* field that specifies a set of subgoals the agent should achieve and the order in which this should happen. Such higher-level clauses also have a *:start* field, but they lack a *:requires* field, which is handled by their primitive subskills. Table 2 shows two nonprimitive skill clauses for achieving *driving-well-in-segment,* which have the same percepts but which have slightly different start conditions and subgoals. The table also includes two skill clauses intended to

```
((driving-well-in-segment ?self ?seg ?lane)
 :percepts  ((self ?self)
             (segment ?seg)
             (lane-line ?lane segment ?seg))
 :relations ((in-segment ?self ?seg)
             (in-lane ?self ?lane)
             (aligned-with-lane-in-segment ?self ?seg ?lane)
             (centered-in-lane ?self ?seg ?lane)
             (steering-wheel-straight ?self)))

((in-rightmost-lane ?self ?clane)
 :percepts  ((self ?self)
             (lane-line ?clane segment ?seg)
             (segment ?seg))
 :relations ((driving-well-in-segment ?self ?seg ?clane)
             (last-lane ?clane)
             (not (lane-to-right ?clane ?anylane))))

((in-lane ?self ?lane)
 :percepts  ((self ?self segment ?seg)
             (lane-line ?lane segment ?seg dist ?dist))
 :tests     ((> ?dist -10) (<= ?dist 0)))

((in-segment ?self ?seg)
 :percepts  ((self ?self segment ?seg)
             (segment ?seg)))
```

*Table 1. Some ICARUS Concepts for In-City Driving, with Variables Indicated by Question Marks.*

achieve the concept *in-segment,* one a primitive variant that invokes the action ***steer* and another that includes a recursive call to itself. Skill clauses for *in-intersection-for-right-turn* have a similar structure, one of which refers to *in-rightmost-lane* as a subgoal, which in turn attempts to achieve *driving-well-in-segment.*

Clearly, both long-term memories are organized in hierarchical terms, with more complex skills and concepts being defined in terms of simpler components. Each hierarchy includes primitive structures at the bottom and specifies increasingly complex structures at higher levels, although direct and indirect recursion can also occur. Most cognitive architectures can model such hierarchical relations, but few raise this notion to a design principle. For example, ACT-R lets production rules specify goals in their left-hand sides and subgoals in their right-hand sides, but the architecture does not require this relationship. Moreover, ICARUS skill clauses refer to concepts in many of their fields, thus providing additional organization on the framework's long-term memories.

ICARUS's short-term belief memory contains instances of defined concepts, which encode specific beliefs about the environment that the agent can infer from its perceptions. Each such instance includes the concept name and objects in the environment that serve as its arguments. For example, this memory might contain the instance *(in-lane self g601),* which could follow from the in-lane concept shown in table 1. Instances of higher-level concepts, such as *(driving-well-in-segment me g550 g601),* also take the form of conceptual predicates with objects as their arguments. Table 3 presents some example beliefs from the in-city driving domain; these clarify that ICARUS's beliefs about its current situation are inherently rela-

```
((driving-well-in-segment ?self ?seg ?line)
 :percepts ((segment ?seg) (lane-line ?line) (self ?self))
 :start    ((steering-wheel-straight ?self))
 :subgoals ((in-segment ?self ?seg)
            (centered-in-lane ?self ?seg ?line)
            (aligned-with-lane-in-segment ?self ?seg ?line)
            (steering-wheel-straight ?self)))

((driving-well-in-segment ?self ?seg ?line)
 :percepts ((segment ?seg) (lane-line ?line) (self ?self))
 :start    ((in-segment ?self ?seg) (steering-wheel-straight ?self))
 :subgoals ((in-lane ?self ?line)
            (centered-in-lane ?self ?seg ?line)
            (aligned-with-lane-in-segment ?self ?seg ?line)
            (steering-wheel-straight ?self)))

((in-segment ?self ?seg)
 :percepts ((self ?self) (intersection ?int) (segment ?seg))
 :start    ((last-lane ?line))
 :subgoals ((in-intersection-for-right-turn ?self ?int)
            (in-segment ?self ?seg)))

((in-segment ?self ?endsg)
 :percepts ((self ?self speed ?speed) (intersection ?int cross ?cross)
            (segment ?endsg street ?cross angle ?angle))
 :start    ((in-intersection-for-right-turn ?self ?int))
 :actions  ((*steer 1)))

((in-intersection-for-right-turn ?self ?int)
 :percepts ((lane-line ?line) (self ?self) (intersection ?int))
 :start    ((last-lane ?line))
 :subgoals ((in-rightmost-lane ?self ?line)
            (in-intersection-for-right-turn ?self ?int)))

((in-intersection-for-right-turn ?self ?int)
 :percepts ((self ?self) (segment ?seg) (intersection ?int)
            (lane-line ?lane segment ?seg))
 :start    ((in-rightmost-lane ?self ?lane))
 :requires ((in-segment ?self ?seg) (intersection-ahead ?int) (last-lane ?lane))
 :actions  ((*cruise)))

((in-rightmost-lane ?self ?line)
 :percepts ((self ?self) (lane-line ?line))
 :start    ((last-lane ?line))
 :subgoals ((driving-well-in-segment ?self ?seg ?line)))
```

*Table 2. Some ICARUS Skills for the In-City Driving Domain.*

```
(current-street me A)                        (current-segment me g550)
(lane-to-right g599 g601)                    (first-lane g599)
(last-lane g599)                             (last-lane g601)
(at-speed-for-u-turn me)                     (slow-for-right-turn me)
(steering-wheel-not-straight me)             (centered-in-lane me g550 g599)
(in-lane me g599)                            (in-segment me g550)
(on-right-side-of-road-in-segment me)        (intersection-behind g550 g522)
(building-on-left g288)                      (building-on-left g425)
(building-on-left g427)                      (building-on-left g429)
(building-on-left g431)                      (building-on-left g433)
(building-on-right g287)                     (building-on-right g279)
(increasing-direction me)                    (buildings-on-right g287 g279)
```

*Table 3. Partial Contents of ICARUS's Short-Term Conceptual Memory for the In-City Driving Domain.*

tional in structure, much as the contents of short-term memories in other architectures like Soar and ACT.

However, ICARUS's perceptual buffer has a somewhat different character. Elements in this memory, which is refreshed on every cycle, describe individual objects that the agent perceives in the environment. Each element has a type (for example, *building* or *segment*), a unique name (for example, *g425*), and a set of attributes with their associated values. Table 4 gives the partial contents of the perceptual buffer for one situation that arises in the in-city driving domain. This includes six lane lines, each of which has a length, width, distance, angle, color, and associated segment. The table also shows four perceived buildings, each with an address, street, distance and angle to the corner closest to the agent that faces the street, and distance and angle to the other corner. Additional percepts describe the agent, road segments, sidewalks, an intersection, and a stoplight.[2] Note that most attributes take on numeric values but that some are symbolic.

ICARUS also incorporates a short-term memory for goals and intentions. This contains a prioritized set of goal stacks, each of which contains an ordered list of goals, with an entry serving as the subgoal for the one below it on the list. Each goal entry may have an associated skill instance that specifies the agent's intention to execute that skill, once it becomes applicable, in order to achieve the goal. Entries may also contain other information about subgoals that have been achieved previously or abandoned. Only the top entry on each goal stack is accessible to the ICARUS interpreter, but older information can become available when the system pops the stack upon achieving or abandoning the current goal.

Unlike other cognitive architectures, ICARUS also imposes a strong correspondence between the contents of its long-term and short-term memories. In particular, it requires that every short-term element be a specific instance of some long-term structure. For example, belief memory contains instances of defined concepts that the agent can infer from its environmental perceptions. Thus, this memory might contain the instance *(in-segment me g550),* which it can infer from the in-segment concept shown in table 1. The same holds for instances that appear in goal memory, in which an element like *(driving-well-in-segment me g550 g601)* indicates the agent's desire to be aligned and centered with respect to a given segment and lane line. In fact, ICARUS cannot encode a goal without a corresponding long-term concept, and the intentions attached to goals must be instances of clauses in long-term skill memory. This suggests a natural approach to modeling episodic traces that my colleagues and I plan to explore in future work.

ICARUS's theoretical position contrasts with those of Soar and ACT-R, which enforce much weaker connections. The latter states that elements in short-term memory are active versions of structures in long-term declarative memory, but makes few claims about the relation between generalized structures and specific instances of them. In both frameworks, production rules in long-term memory contain generalized patterns that match or alter specific elements in short-term memory, but ICARUS's relationship is more constrained. On this dimension, ICARUS comes closer to Schank's (1982)

```
(self me speed 5 angle-of-road -0.5 steering-wheel-angle -0.1)
(segment g562 street 1 dist -5.0 latdist 15.0)
(lane-line g564 length 100.0 width 0.5 dist 35.0 angle 1.1 color white segment g562)
(lane-line g565 length 100.0 width 0.5 dist 15.0 angle 1.1 color white segment g562)
(lane-line g563 length 100.0 width 0.5 dist 25.0 angle 1.1 color yellow segment g562)
(segment g550 street A dist oor latdist nil)
(lane-line g6OO length 100.0 width 0.5 dist -15.0 angle -0.5 color white segment g550)
(lane-line g6O1 length 100.0 width 0.5 dist 5.0 angle -0.5 color white segment g550)
(lane-line g599 length 100.0 width 0.5 dist -5.0 angle -0.5 color yellow segment g550)
(intersection g522 street A cross 1 dist -5.0 latdist nil)
(building g431 address 99 street A c1dist 38.2 c1angle -1.4 c2dist 57.4 c2angle -1.0)
(building g429 address 74 street A c1dist 29.0 c1angle -2.1 c2dist 38.2 c2angle -1.4)
(building g425 address 25 street A c1dist 37.8 c1angle -2.8 c2dist 56.9 c2angle -3.1)
(building g389 address 49 street 1 c1dist 49.2 c1angle 2.7 c2dist 53.0 c2angle 2.2)
(sidewalk g471 dist 15.0 angle -0.5)
(sidewalk g474 dist 5.0 angle 1.07)
(sidewalk g469 dist -25.0 angle -0.5)
(sidewalk g470 dist 45.0 angle 1.07)
(stoplight g538 vcolor green hcolor red))
```

*Table 4. Partial Contents of ICARUS's Perceptual Buffer for the In-City Driving Domain.*

theory of dynamic memory, which does not meet all of the criteria for a cognitive architecture but which he proposed in much the same spirit. In addition, both frameworks share a commitment to the hierarchical organization of memory and reserve a central role for conceptual inference.

## Performance and Learning Processes

Besides making theoretical claims about memories and their contents' representations, a cognitive architecture also commits to a set of processes that alter these contents. These are described at the level of functional mechanisms, which is more concrete than Newell's (1982) "knowledge level" and more abstract than the implementation level of hardware or wetware. Thus, the architecture specifies each process in terms of an algorithm or procedure that is independent of its implementation details, yet still operates over particular mental structures.

Research on cognitive architectures, like psychology, generally distinguishes between *performance* processes and *learning* processes. Performance mechanisms utilize structures in long-term memory to interpret and alter the contents of short-term memory, making them responsible for the generation of beliefs and goals. These typically include methods for memory retrieval, pattern matching, skill selection, inference, and problem solving. In contrast, learning processes are responsible for altering the contents of long-term memory, either by generating new knowledge structures or by refining and modulating existing structures. In most architectures, the mechanisms for performance and learning are closely intertwined.

For example, figure 2 indicates that ICARUS includes separate performance modules for conceptual inference, skill execution, and problem solving, but they operate on many of the same structures and they build on each others' results in important ways. In particular, the problem-solving process is interleaved with skill retrieval and execution, and both rely heavily on beliefs produced by the inference module to determine their behavior. Furthermore, the hierarchical organization of long-term memory plays a central role in each of their mechanisms.

Conceptual inference is the architecture's most basic activity. On each cycle, the system matches concept definitions in long-term memory against perceptions and beliefs. When a concept matches, the module adds an instance of that concept to short-term belief memory, making it available to support other inferences. As the left side of figure 3 depicts, the system operates in a bottom-up manner, starting with primitive concepts, which match against percepts, and working up to higher-level concepts, which match against lower-level

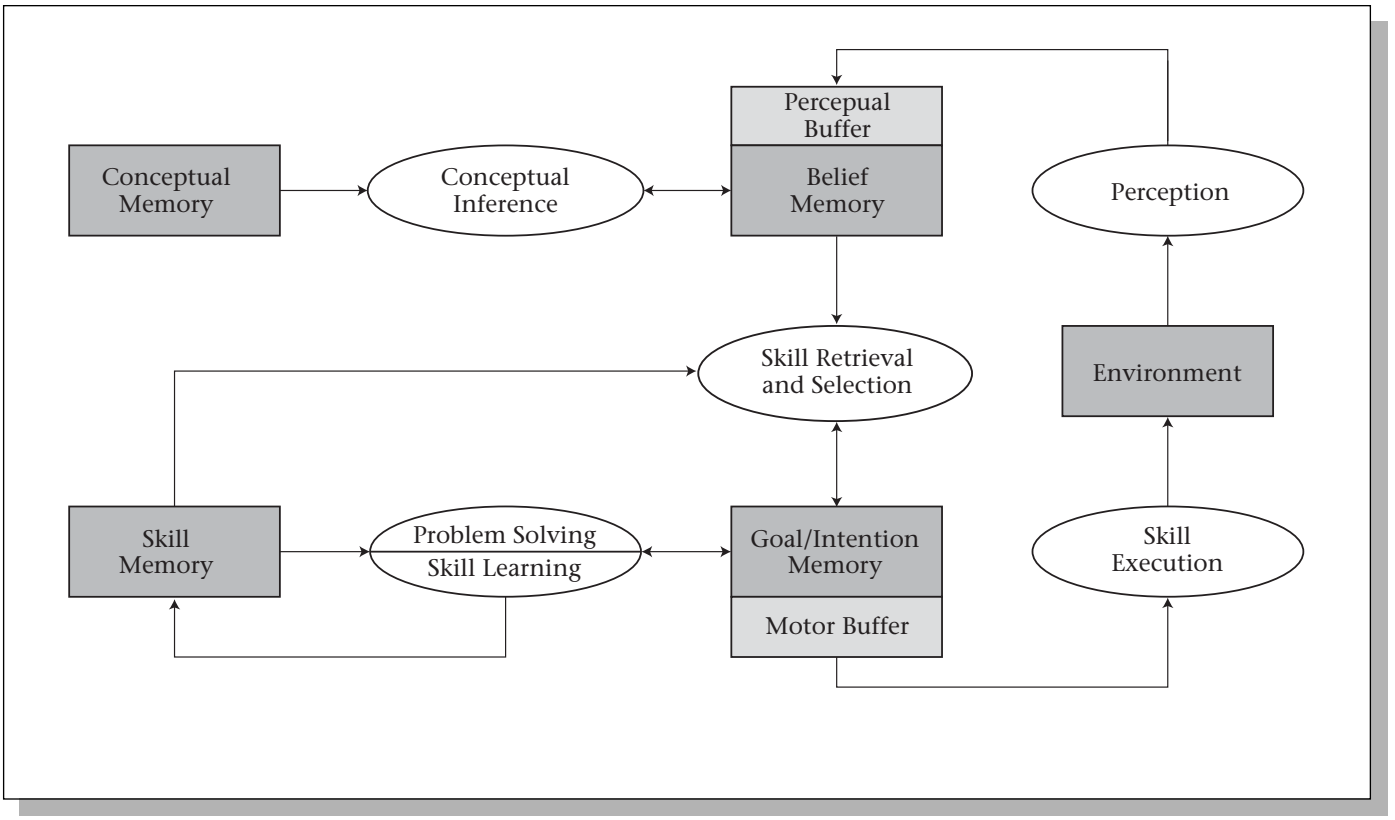


*Figure 2. Functional Processes of the ICARUS Architecture and Their Connections to Memories.*

concepts. This cascade continues until ICARUS has deduced all beliefs that are implied by its conceptual knowledge base and by its immediate perceptions.

In contrast, the skill execution module proceeds in a top-down manner, as the right side of figure 3 illustrates. The process starts from the current goal, such as *(on-street me A)* or *(driving-well-in-segment me ?seg ?lane),* and finds applicable paths through the hierarchy that terminate in primitive skill clauses with executable actions, such as *(*steer 1)*. A skill path is a chain of skill instances that starts from the agent's top-level goal and descends the skill hierarchy, unifying the arguments of each subskill clause consistently with those of its parent. A path is *applicable* if the concept instance that corresponds to the intention is not satisfied, if the requirements of the terminal (primitive) skill instance are satisfied, and if, for each skill instance in the path not executed on the previous cycle, the start conditions are satisfied. This last constraint is necessary because skills may take many cycles to achieve their desired effects, making it important to distinguish between their initiation and their continuation.

When ICARUS's execution module can find a path through the skill hierarchy relevant to its current goal, it carries out actions in the environment, but when it cannot find such a path, it invokes a module for means-ends problem solving (Newell and Simon 1961). This chains backward from the goal off either a skill or concept definition, pushing the result of each reasoning step onto a goal stack. The module continues pushing new goals onto the stack until it finds one it can achieve with an applicable skill, in which case it executes the skill and pops the goal from the stack. If the parent goal involved skill chaining, then this leads to execution of its associated skill and achievement of the parent, which is in turn popped. If the parent goal involved concept chaining, another unsatisfied subconcept is pushed onto the goal stack or, if none remain, then the parent is popped. This process continues until the system achieves the top-level goal.

ICARUS's performance processes have clear similarities to analogous mechanisms in other architectures. Conceptual inference plays much the same role as the elaboration stage in Soar, which adds inferences to short-term memory in a deductive, bottom-up manner for use in decision making. Selection and execution of skill paths bears a strong resemblance to

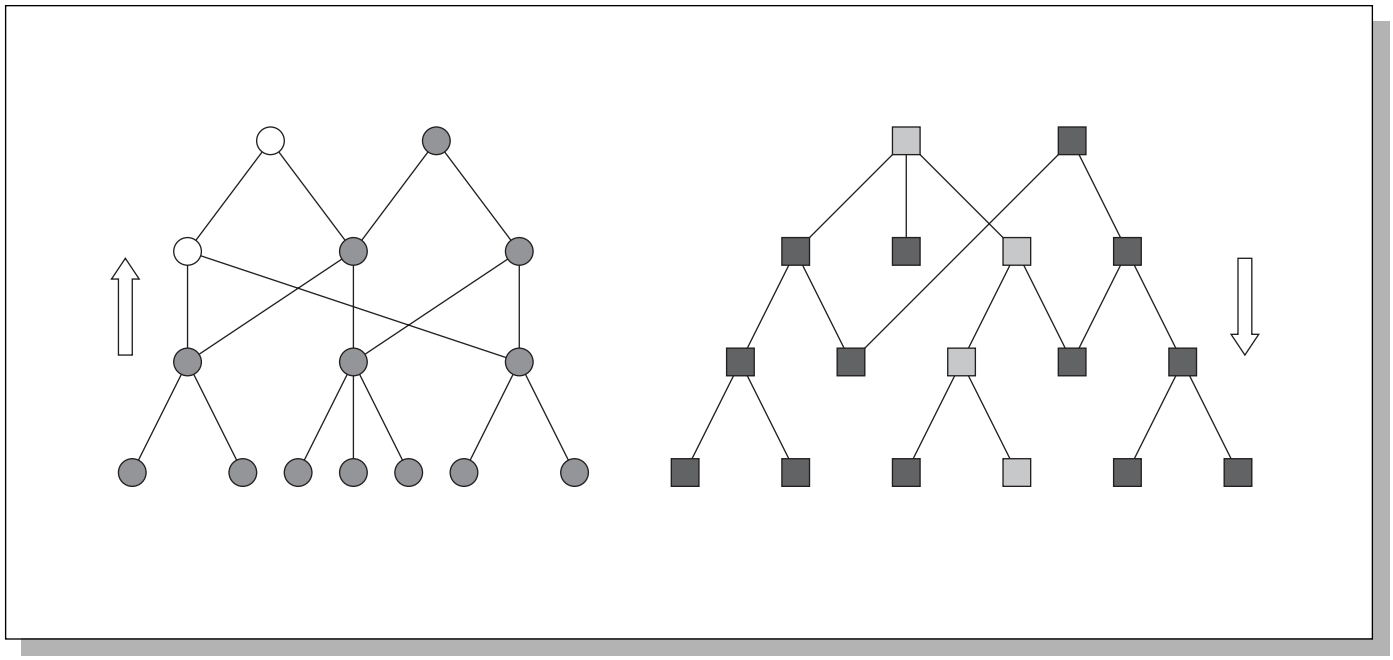

*Figure 3. ICARUS Conceptual Clauses (left) Are Matched Bottom Up, Starting from Percepts, Whereas Skill Clauses (right) Are Matched Top Down, Starting from the Agent's Goals.*

On the left, darker nodes denote matched concept instances; on the right, lighter nodes indicate an applicable path through the skill hierarchy.

the goal-driven, top-down control typically utilized in ACT-R systems, although ICARUS uses this idea for executing physical actions rather than cognitive processing, and it traverses many levels of the skill hierarchy on each decision cycle. The means-ends problem solver operates much like the one central to Prodigy (Minton et al. 1989), except that it interleaves planning with execution, which reflects ICARUS's commitment to embedding cognition in physical agents.

Finally, ICARUS incorporates a learning module that creates a new skill whenever problem solving and execution achieve a goal. The new structure includes the achieved goal as its head, the subgoals that led to the goal as its subskills, and start conditions that differ depending on whether the solution involved chaining off a skill or concept definition. As discussed in more detail elsewhere (Langley and Choi, 2006), learning is interleaved with problem solving and execution, and it occurs in a fully incremental manner. Skills acquired earlier are available for inclusion in those formed later, making the learning process cumulative. ICARUS shares with Soar and Prodigy the notion of learning from impasses that are overcome through problem solving, but it differs in its ability to acquire hierarchical skills in a cumulative fashion that builds on earlier structures.

I should reiterate that ICARUS's various modules do not proceed in an independent fashion but are constrained by each others' operation. Skill execution is influenced by the inference process because the former tests against concept instances produced by the latter. Problem solving is constrained both by execution, which it uses to achieve subgoals, and by inference, which lets it determine when they have been achieved. Finally, skill learning draws directly on the results of problem solving, which let it determine the structure of new skills, and inferred beliefs, which determine the start conditions it should place on these skills. Such strong interaction is the essence of a cognitive architecture that aspires to move beyond integration to a unified theory of intelligence.

## Architectures as Programming Languages

Finally, I should note that a cognitive architecture typically comes with an associated programming language for use in building knowledge-based systems. The syntax of this formalism is linked closely to the framework's representational assumptions, with knowledge in long-term memory corresponding to the program and with initial short-term elements playing the role of inputs. The language includes an interpreter that can run the program

on these inputs, and usually comes with tracing facilities that let users inspect the system's behavior over time.

In general, the languages associated with cognitive architectures are higher level than traditional formalisms, letting them produce equivalent behavior with much more concise programs. This power comes partly from the architecture's commitment to specific representations, which incorporate ideas from list processing and first-order logic, but it also follows from the inclusion of processes that interpret these structures in a specific way. Mechanisms like pattern matching, inference, and problem solving provide many implicit capabilities that must be provided explicitly in traditional languages. For these reasons, cognitive architectures support far more efficient development of software for intelligent systems, making them the practical choice for many applications.

The programming language associated with ICARUS comes with the syntax for hierarchical concepts and skills, the ability to load and parse such programs, and commands for specifying the initial contents of short-term memories and interfaces with the environment. The language also includes an interpreter that handles inference, execution, planning, and learning over these structures, along with a trace package that displays system behavior on each cycle. I have presented examples of the syntax and discussed the operation of the interpreter in previous sections, and my colleagues and I have used this language to develop adaptive intelligent agents in a variety of domains.

As noted earlier, the most challenging of these has involved in-city driving. For example, we have constructed an ICARUS program for delivering packages within the simulated driving environment that includes 15 primitive concepts and 55 higher-level concepts, which range from one to six levels deep. These are grounded in perceptual descriptions for buildings, road segments, intersections, lane lines, packages, other vehicles, and the agent's vehicle. The system also incorporates eight primitive skills and 33 higher-level skills, organized in a hierarchy that is five levels deep. These terminate in executable actions for changing speed, altering the wheel angle, and depositing packages. We have used this domain to demonstrate the integration of conceptual inference, skill execution, problem solving, and acquisition of hierarchical skills.

There also exist other, more impressive, examples of integrated intelligent systems developed within more established cognitive architectures. For instance, Tambe et al. (1995) report a simulated fighter pilot, implemented within the Soar framework, that incorporates substantial knowledge about flying missions and that has been used repeatedly in large-scale military training exercises. Similarly, Trafton et al. (2005) describe an ACT-R system, which controls a mobile robot that interacts with humans in building environments having obstacles and occlusion. These developers have not compared directly the lines of code required to program such systems within a cognitive architecture and within traditional programming languages. However, I am confident that the higher-level constructs available in ICARUS, Soar, ACT-R, and similar frameworks allow much simpler programs and far more rapid construction of intelligent agents.

## Concluding Remarks

In the preceding pages, I reviewed the notion of a cognitive architecture and argued for its role in developing general intelligent systems that have the same range of abilities as humans. I also examined one such architecture—ICARUS—in some detail. My purpose was to illustrate the theoretical commitments made by cognitive architectures, including their statements about system memories, the representation of those memories' contents, and the functional processes that operate on those contents. I also showed how, taken together, these assumptions can support a programming language that eases the construction of intelligent agents.

I should reiterate that ICARUS is neither the most mature or most widely used framework of this sort. Both ACT-R (Anderson 1993) and Soar (Laird, Newell, and Rosenbloom 1987) are many years older and have been used by far more people. Other well-known but more recent cognitive architectures include EPIC (Kieras and Meyer 1997) and Clarion (Sun, Merrill, and Peterson 2001). I will not attempt to be exhaustive here, since research in this area has been ongoing since the 1970s, and I can only hope to mention a representative sample of this important intellectual movement.

However, I should note that the great majority of research efforts on cognitive architectures, including those just mentioned, have focused on production systems, in which condition-action rules in long-term memory match against and modify elements in short-term memory. This paradigm has proven quite flexible and successful in modeling intelligent behavior, but this does not mean the space of cognitive architectures lacks other viable candidates. For reasons given earlier, I view ICARUS as occupying a quite different region of this space, but it shares features with Minton et al.'s Prodigy, which uses means-ends analysis to direct learning, and Freed's (1998) APEX, which stores complex skills in a hierarchical manner. Yet the space is large, and we need more systematic exploration of alternative frameworks that support general intelligent systems.

The field would also benefit from increased research on topics that have received little attention within traditional cognitive architectures. For instance, there has been considerable effort on procedural memory, but much less on episodic memory, which supports quite different abilities. Also, most architectural research has focused on generating the agent's own behavior, rather than on understanding the actions of others around it, which is equally important. Nor do many current cognitive architectures explain the role that emotions might play in intelligent systems, despite their clear importance to human cognition. These and many other issues deserve fuller attention in future research.

Of course, there is no guarantee that work on unified cognitive architectures will lead to computational systems that exhibit human-level intelli-

gence. However, recall that, to date, we have only one demonstration that such systems are possible—humans themselves—and most research on cognitive architectures, even when it does not attempt to model the details of human behavior, is strongly influenced by psychological findings. At the very least, studies of human cognition are an excellent source of ideas for how to build intelligent artifacts, and most cognitive architectures already incorporate mechanisms with such origins. Combined with the aim of developing strong theories of the mind and the desire to demonstrate broad generality, this emphasis makes cognitive architectures a viable approach to achieving human-level intelligence.

## Acknowledgements

This material is based on research sponsored by DARPA under agreement numbers HR0011-04-1-0008 and FA8750-05-2-0283 and by Grant IIS-0335353 from the National Science Foundation. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and do not necessarily represent the official policies or endorsements, either expressed or implied, of DARPA or the U.S. government. Discussions with John Anderson, Randolph Jones, John Laird, Allen Newell, David Nicholas, Stellan Ohlsson, and Stephanie Sage contributed to many of the ideas presented in this article. Dongkyu Choi, Seth Rogers, and Daniel Shapiro have played central roles in the design and implementation of ICARUS, with the former developing the driving agent I have used as my central example.

## Notes

1. An important form of long-term storage—episodic memory—has received remarkably little attention within the cognitive architecture community, although Nuxoll and Laird (2004) report some recent efforts in this area.

2. Of course, we might have modeled the results of perception at a finer granularity, say at the level of object surfaces or edges, but the current architecture is agnostic about such issues.

## References

Anderson, J. R. 1993. *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum.

Byrne, M. D. 2001. ACT-R/PM and Menu Selection: Applying a Cognitive Architecture to HCI. *International Journal of Human-Computer Studies* 55(1): 41–84.

Choi, D.; Kaufman, M.; Langley, P.; Nejati, N.; and Shapiro, D. 2004. An Architecture for Persistent Reactive Behavior. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems,* 988–995. New York: ACM Press.

Clocksin, W. F.; and Mellish, C. S. 1981. *Programming in Prolog*. Berlin: Springer-Verlag.

Engelmore, R. S.; and Morgan, A. J., eds. 1989. *Blackboard Systems*. Reading, MA: Addison-Wesley.

Freed, M. 1998. Managing Multiple Tasks in Complex, Dynamic Environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence,* 921–927. Menlo Park, CA: AAAI Press.

Kieras, D.; and Meyer, D. E. 1997. An Overview of the Epic Architecture for Cognition and Performance with Application to Human-Computer Interaction. *Human-Computer Interaction* 12(4): 391–438.

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. 1990. Integrating Execution, Planning, and Learning in Soar for External Environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence,* 1022–1029. Menlo Park, CA: AAAI Press.

Langley, P.; and Choi, D. 2006. Learning Recursive Control Programs from Problem Solving. *Journal of Machine Learning Research*. 7: 493–518.

Minton, S.; Carbonell, J. G.; Knoblock, C. A.; Kuokka, D.; Etzioni, O.; and Gil, Y. 1989. Explanation-Based Learning: A Problem Solving Perspective. *Artificial Intelligence* 40(1–3): 63–118.

Newell, A. 1973. You Can't Play 20 Questions with Nature and Win: Projective Comments on the Papers of This Symposium. In *Visual Information Processing* ed. W. G. Chase. New York: Academic Press.

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

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

Newell, A.; and Simon, H. A. 1961. GPS, A Program That Simulates Human Thought. In *Lernende Automaten,* ed. H. Billing. Munich: Oldenbourg KG. Reprinted in *Computers and Thought*, ed. E. A. Feigenbaum and J. Feldman. New York: McGraw-Hill, 1963.

Newell, A.; and Simon, H. A. 1976. Computer Science as Empirical Enquiry: Symbols and Search. *Communications of the ACM* 19(3): 113–126.

Nuxoll, A.; and Laird, J. E. 2004. A Cognitive Model of Episodic Memory Integrated with a General Cognitive Architecture. In *Proceedings of the Sixth International Conference on Cognitive Modeling,* 220–225. Mahwah, NJ: Lawrence Erlbaum.

Schank, R. C. 1982. *Dynamic Memory*. Cambridge, U.K.: Cambridge University Press.

Sun, R.; Merrill, E.; and Peterson, T. 2001. From Implicit Skills to Explicit Knowledge: A Bottom-Up Model of Skill Learning. *Cognitive Science* 25(2): 203–244.

Sycara, K. 1998. Multi-Agent Systems. *AI Magazine* 19(2): 79–93.

Tambe, M.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. B. 1995. Intelligent Agents for Interactive Simulation Environments. *AI Magazine* 16(1): 15–39.

Trafton, J. G.; Cassimatis, N. L.; Bugajska, M.; Brock, D.; Mintz, F.; and Schultz, A. 2005. Enabling Effective Human-Robot Interaction Using Perspective-Taking in Robots. *IEEE Transactions on Systems, Man and Cybernetics* 25(4): 460–470.

**Pat Langley** serves as the Director of the Institute for the Study of Learning and Expertise, Consulting Professor of Symbolic Systems at Stanford University, and Head of the Computational Learning Laboratory at Stanford's Center for the Study of Language and Information. He has contributed to the fields of artificial intelligence and cognitive science for more than 25 years, having published 200 papers and five books on these topics including *Elements of Machine Learning*. Langley is a AAAI Fellow. He was a founding editor of the journal *Machine Learning*. He was program chair for the Seventeenth International Conference on Machine Learning. His research has dealt with learning in planning, reasoning, language, vision, robotics, and scientific knowledge discovery, and he has contributed novel learning methods to the logical, probabilistic, and case-based paradigms. His current research focuses on methods for constructing explanatory process models in scientific domains and on cognitive architectures for physical agents.