Cognitive Architectures and General Intelligent Systems

Pat Langley
Computational Learning Laboratory
Center for the Study of Language and Information
Stanford University, Stanford, CA 94305

1. 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. We 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 claimed that we should demonstrate our systems' intelligence on the same range of domains and tasks as handled by humans, and that we 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 we can best explain by contrasting it with alternative frameworks.

2. 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 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 & 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, the architecture of a blackboard system supports a different form of integration than multi-agent scheme that 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 multi-agent 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:

- the short-term and long-term memories that store the agent's beliefs, goals, and knowledge;
- the representation and organization of structures that are embedded in these memories;
- the functional processes that operate on these structures, including both performance and learning mechanisms;
- a programming language that lets one construct knowledge-based systems which 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, we will use one such cognitive architecture – Icarus – to illustrate each of these commitments in turn. Icarus is neither the oldest or most developed architecture; some frameworks, like ACT (Anderson, 1993) and Soar (Laird, Newell, & Rosenbloom, 1987), have undergone continual development for over 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, we 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. 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, 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 our main points.

3. 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 knowledge 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 six high-level principles about the nature of general intelligent systems:

- cognition is grounded in perception and action;
- concepts and skills are distinct cognitive structures;
- long-term memory is organized in an hierarchical fashion;
- skill and concept hierarchies are acquired in a cumulative manner;
- long-term and short-term structures have a strong correspondence;
- symbolic cognitive structures are modulated with numeric functions.[1]

These ideas further distinguish Icarus from most other cognitive architectures that have been developed within the Newell tradition. Again, we will not claim here that they make our framework superior to earlier ones, but we believe they do clarify the dimensions that define the space of candidate architectures.

3.1 Memories and Representations

To reiterate, a cognitive architecture makes a commitment to the memories which store the content that control its behavior. These must include one or more long-term memories that contain

1. Unlike Icarus' other features, we will not discuss this issue further in the current paper, but we have considered its role at more length elsewhere (Choi et al., 2004).

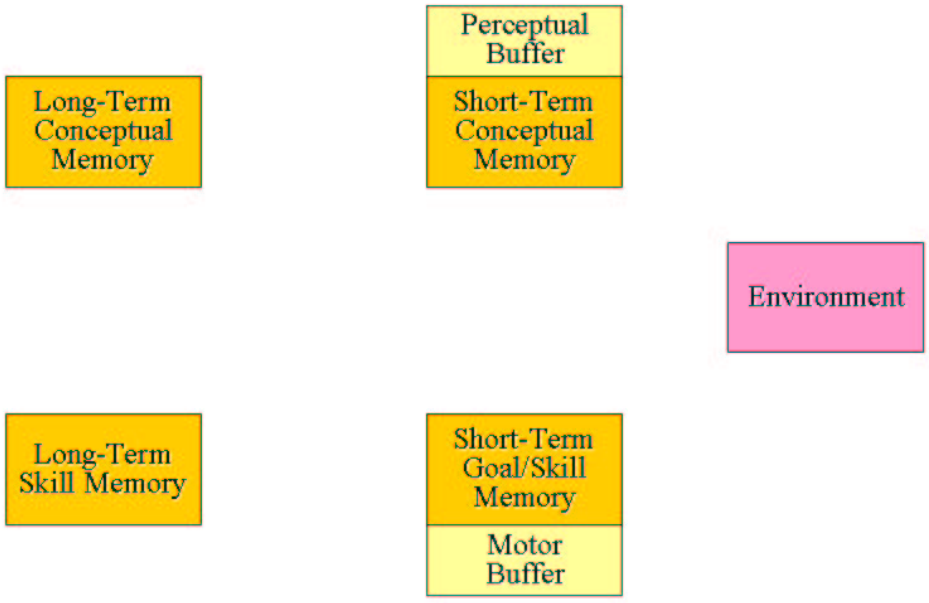

Figure 1. Six long-term and short-term memories of the Icarus architecture.

knowledge and procedures, along with one or more short-term meories that store the agent's beliefs and goals. The contents of long-term memories change gradually or not at all, whereas short-term elements change rapidly in response to environmental conditions and the agent's agenda. Some architectures also incorporate sensori-motor 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' 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

Table 1. Some ICARUS concepts for in-city driving, with variables indicated by question marks.

```
(in-segment (?self ?sg)
 :percepts  ((self ?self segment ?sg) (segment ?sg)) )
(aligned-with-lane (?self ?lane)
 :percepts  ((self ?self) (lane-line ?lane angle ?angle))
 :positives ((in-lane ?self ?lane))
 :tests     ((> ?angle -0.05) (< ?angle 0.05)) )
(on-street (?self ?packet)
 :percepts  ((self ?self) (packet ?packet street ?street)
             (segment ?sg street ?street))
 :positives ((not-delivered ?packet)
             (current-segment ?self ?sg)) )
(increasing-direction (?self)
 :percepts  ((self ?self))
 :positives ((increasing ?b1 ?b2))
 :negatives ((decreasing ?b3 ?b4)) )
```

similar to the predicate calculus that support the expression of relational content. These build on AI's central assumption that intelligent behavior involves the manipulation of list structures. 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 observed perceptual entities, a *:positives* field that states lower-level concepts it must match, a *:negatives* field that gives concepts it must not match, and a *:tests* field that specifies numeric relations it must satisfy. Table 1 shows some concepts from the driving domain that illustrate both primitive concepts (e.g., *in-segment*) and nonprimitive ones (e.g., *on-street* and *increasing-direction*).

In contrast, long-term skill memory encodes knowledge about ways to act and achieve goals. Each skill has a head, which gives its name and arguments, and a body with a variety of fields. For primitive skills, these include an *:effects* field that specifies concepts the skill is intended to achieve, a *:start* field that describes the situation in which one can initiate the skill, a *:requires* field that must hold throughout the skill's execution, and an *:actions* field that indicates executable actions the skill should invoke. For example, Table 2 shows the primitive skill *steer-for-right-turn*, which makes the *in-segment* concept true and which is considered only when *ready-for-right-turn* holds.

Nonprimitive skills differ from primitive ones in that they have no *:actions* field, since they instead have a *:skills* field that specifies a set of subskills the agent should execute and the order in which they should occur. Such higher-level skills also have a *:start* field, but they lack a *:requires* field, which is handled by their primitive subskills, and an *:effects* field, which is encoded by the literals

Table 2. Primitive and nonprimitive ICARUS skills for the in-city driving domain.

```
(on-street-right-direction (?self ?packet)
 :percepts ((self ?self segment ?segment direction ?dir)
            (building ?landmark))
 :start    ((on-street-wrong-direction ?self ?packet))
 :ordered  ((get-in-U-turn-lane ?self)
            (prepare-for-U-turn ?self)
            (steer-for-U-turn ?self ?landmark)) )

(get-aligned-in-segment (?self ?sg)
 :percepts ((lane-line ?lane angle ?angle))
 :requires ((in-lane ?self ?lane))
 :actions  ((*steer (*times ?angle 2)))
 :effects  ((aligned-with-lane ?self ?lane)) )

(steer-for-right-turn (?self ?int ?endsg)
 :percepts ((self ?self speed ?speed)
            (intersection ?int cross ?cross)
            (segment ?endsg street ?cross angle ?angle))
 :start    ((ready-for-right-turn ?self ?int))
 :actions  ((*steer (*times ?angle 2)))
 :effects  ((in-segment ?self ?endsg)) )
```

in their heads. Table 2 shows the nonprimitive skill *on-street-right-direction*, which refers to the concept *on-street-wrong-direction* in its *:start* field and has three ordered subskills.

Clearly, both long-term memories are organized in hierarchical terms, with more complex skills and concepts being defined in terms of simpler components. Both hierarchies include primitive structures at the bottom and specify increasingly complex structures at higher levels. Most cognitive architectures can model such hierarchical relations, but few raise this notion to a design principle. For example, ACT-R lets production rules link goals to subgoals, but the relation remains mediated by working memory elements rather than referring directly to component structures. Moreover, ICARUS skills refer to concepts in some of their fields, thus providing additional organization on the framework's long-term memories.

ICARUS' short-term conceptual 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-segment self g0037)*, which could follow from the *in-segment* concept shown in Table 1. Instances of higher-level concepts take the same form of conceptual predicates with objects as their arguments. Thus, ICARUS' beliefs about its current situation are inherently relational in structure, as the examples in Table 3 illustrate.

The 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 (e.g., *building* or *segment*), a unique name (e.g., *g0019*), and a set of

Table 3. Partial contents of ICARUS' short-term conceptual memory for the in-city driving domain.

(buildings-on-right me g2231 g2230 g2480)	(increasing me g2231 g2230 g2480)
(buildings-on-right me g2231 g2222 g2480)	(increasing me g2231 g2222 g2480)
(buildings-on-right me g2231 g2211 g2480)	(increasing me g2231 g2211 g2480)
(buildings-on-right me g2230 g2222 g2480)	(increasing me g2230 g2222 g2480)
(buildings-on-right me g2230 g2211 g2480)	(increasing me g2230 g2211 g2480)
(buildings-on-right me g2222 g2211 g2480)	(increasing me g2222 g2211 g2480)
(buildings-on-left me g2366 g2480)	(buildings-on-left me g2368 g2480)
(buildings-on-left me g2370 g2480)	(buildings-on-left me g2372 g2480)
(not-on-street me g2980)	(currrent-building me g2222)
(not-approaching-cross-street me g2980)	(not-on-cross-street me g2980)
(current-street me A)	(current-segment me g2480)
(not-delivered g2980)	(in-U-turn-lane me g2533)
(in-leftmost-lane me g2533)	(lane-to-right me g2533)
(fast-for-right-turn me)	(fast-for-U-turn me)
(driving-in-segment me g2480 g2533)	(at-speed-for-cruise me)
(steering-wheel-straight me)	(centered-in-lane me g2533)
(aligned-with-lane me g2533)	(in-lane me g2533)
(on-right-side-of-road me)	(in-segment me g2480)

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. Note that most attributes take on numeric values but that some are symbolic. 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.

ICARUS also incorporates a short-term memory for goals and intentions. This contains a set of goal stacks, each of which contains an ordered list of goals, with each entry serving as the subgoal for the one below it on 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 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, short-term conceptual memory contains instances of defined concepts which encode specific beliefs about the environment that the agent can infer from its perceptions. Thus, this memory might contain the instance *(in-segment me g2480)*, which it can infer from the *in-segment* concept shown in Table 4. The same holds for instances that appear in the short-term goal memory, in which an element like *(on-street-right-direction me g2480)* indicates the agent's desire to be driving in a direction that takes it

Table 4. Partial contents of Icarus' perceptual buffer for the in-city driving domain.

```
(self me speed 24.0 wheel-angle 0.02 limit 25.0 road-angle 0.06)
(segment g1059 street 2 dist -5.0 latdist 15.0)
(segment g1050 street A dist -45.0 latdist nil)
(segment g1049 street A dist oor latdist nil)
(lane-line g1073 length 100.0 width 0.5 dist 35.0 angle 1.57 color white)
(lane-line g1074 length 100.0 width 0.5 dist 15.0 angle 1.57 color white)
(lane-line g1072 length 100.0 width 0.5 dist 25.0 angle 1.57 color yellow)
(lane-line g1100 length 100.0 width 0.5 dist -15.0 angle 0.0 color white)
(lane-line g1101 length 100.0 width 0.5 dist 5.0 angle 0.0 color white)
(lane-line g1099 length 100.0 width 0.5 dist -5.0 angle 0.0 color yellow)
(lane-line g1104 length 100.0 width 0.5 dist 5.0 angle 0.0 color white)
(intersection g1021 street A cross 2 dist -5.0 latdist nil)
(building g943 address 246 c1dist 43.69 c1angle -0.73 c2dist nil c2angle nil)
(building g941 address 246 c1dist 30.10 c1angle -1.30 c2dist 43.70 c2angle -0.73)
(building g939 address 197 c1dist 30.10 c1angle -1.30 c2dist 33.40 c2angle -2.10)
(building g943 address 172 c1dist 33.40 c1angle -2.09 c2dist 50.39 c2angle -2.53)
(sidewalk g975 dist 15.0 angle 0.0)
(sidewalk g978 dist 5.0 angle 1.57)
```

closer to the address specified on package *g2480*. In fact, Icarus cannot encode a goal without a corresponding long-term concept. Similarly, the intentions attached to goals must be instances of skills stored in long-term skill memory.

This 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 no 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' relationship is far more constrained. On this dimension, Icarus comes closer to Schank's (1982) theory of dynamic memory, which does not meet all of our criteria for a cognitive architecture but which he proposed in much the same spirit.

3.2 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.

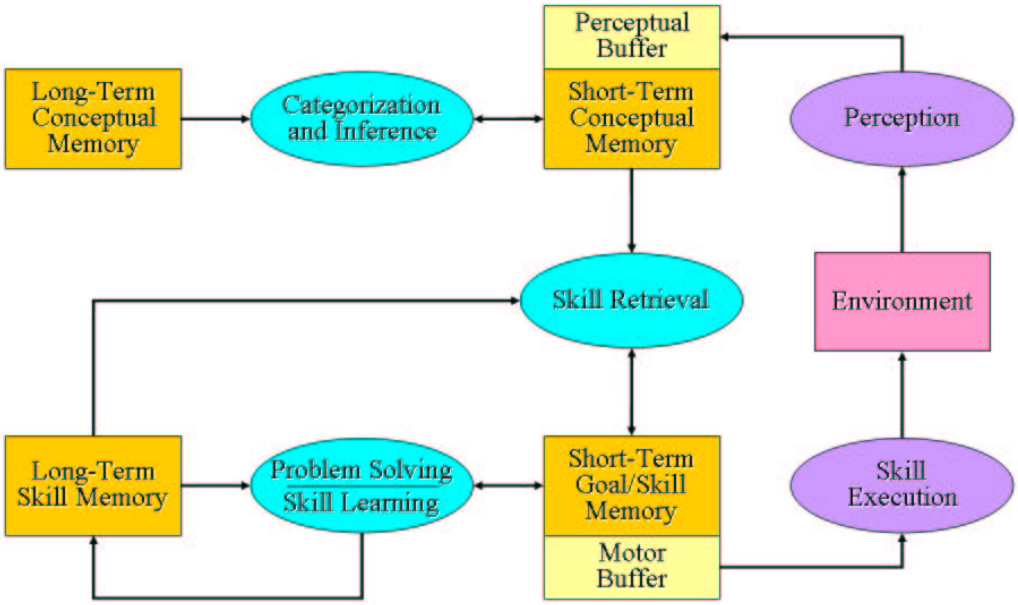

Figure 2. Functional processes of Icarus architecture and their connections to memories.

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

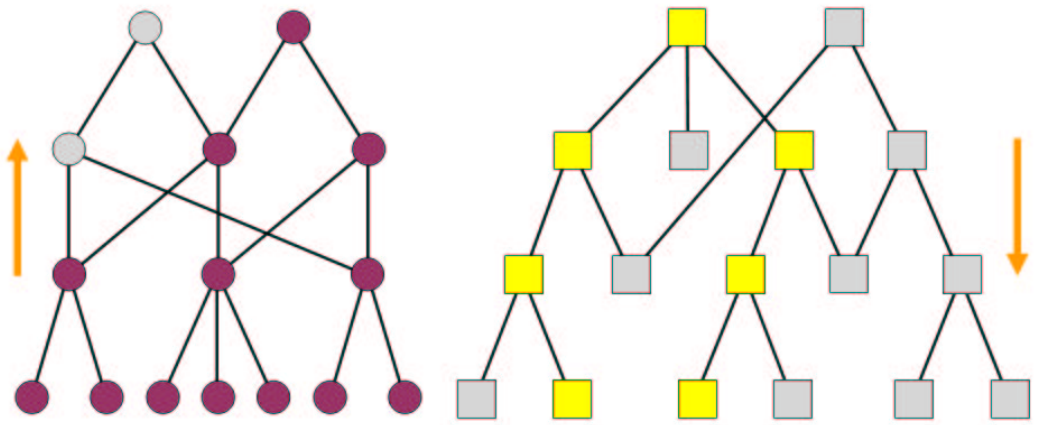

Figure 3. Icarus concepts are matched bottom up, starting from percepts, whereas skills are matched top down, starting from the agent's goals.

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 g2980)* or *(centered-in-lane me g2533)*, and finds applicable paths through the hierarchy that terminate in primitive skills with executable actions, such as *(*steer (*times ?angle 2))*. 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 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' 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. 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 subconcepts 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' 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 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' 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 we discuss in more detail elsewhere (Choi & Langley, 2005), 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.

We should reiterate that Icarus' 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.

3.3 Architectures as Programming Languages

Finally, we 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. We have presented examples of the syntax and discussed the operation of the interpreter in previous sections, and we 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, we are confident that the higher-level constructs available in Icarus, Soar, and ACT-R allow much simpler programs and far more rapid construction of intelligent agents.

4. Concluding Remarks

In the preceding pages, we 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. We also examined one such architecture – Icarus – in some detail. Our 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. We also showed how, taken together, these assumptions can support a programming language that eases the construction of intelligent agents.

We 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 et al., 1987) are many years older and have been used by far more users. Other well-known but more recent cognitive architectures include EPIC (Kieras & Meyer, 1997) and Clarion (Sun, Merrill, & Peterson, 2001). We will not attempt to be exhaustive here, since research in this area has been ongoing since the 1970s, and we can only hope to mention a representative sample of this important intellectual movement.

However, we should note that the great majority of research on cognitive architectures, including those just mentioned, has 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, we 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 intelligence. However, we should 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 research was funded in part by Grant IIS-0335353 from the National Science Foundation and by Grant HR0011-04-1-0008 from Rome Labs. Discussions with John Anderson, Randy Jones, John Laird, Allen Newell, David Nicholas, Stellan Ohlsson, and Stephanie Sage contributed to many of the ideas presented in this paper. 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 we have used as our central example.

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*, 41–84.

Choi, D., Kaufman, M., Langley, P., Nejati, N., & Shapiro, D. (2004). An architecture for persistent reactive behavior. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems* (pp. 988–995). New York: ACM Press.

Choi, D., & Langley, P. (2005). Learning teleoreactive logic programs from problem solving. *Proceedings of the Fifteenth International Conference on Inductive Logic Programming* (pp. 51–68). Bonn, Germany: Springer.

Engelmore, R. S., & Morgan, A. J., (Eds.) (1989)., *Blackboard systems*. Reading, MA: Addison-Wesley.

Freed, M. (1998). Managing multiple tasks in complex, dynamic environments. *Proceedings of the National Conference on Artificial Intelligence* (pp. 921–927). Madison, WI: AAAI Press.

Kieras, D., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, *12*, 391–438.

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

Laird, J. E., & Rosenbloom, P. S. (1990). Integrating execution, planning, and learning in Soar for external environments. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 1022–1029). Boston, MA: AAAI Press.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.) *Visual information processing*. New York: Academic Press.

Newell A. (1982). The knowledge level. *Artificial Intelligence*, *18*, 87–127.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Schank, R. C. (1982). *Dynamic memory*. Cambridge: Cambridge University Press.

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom-up model of skill learning. *Cognitive Science*, *25*, 203–244.

Sycara, K. (1998) Multi-agent systems. *AI Magazine*, *10*, 79–93.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, *16*, 15–39.

Trafton, J. G., Cassimatis, N. L., Bugajska, M., Brock, D., Mintz, F., & Schultz, A., (2005). Enabling effective human-robot interaction using perspective-taking in robots. *IEEE Transactions on Systems, Man and Cybernetics*, *25*, 460–470.