An Adaptive Architecture for Physical Agents

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

Abstract

In this paper we describe ICARUS*, an adaptive architecture for intelligent physical agents. We contrast the framework's assumptions with those of earlier architectures, taking examples from an in-city driving task to illustrate our points. Key differences include primacy of perception and action over problem solving, separate memories for categories and skills, a hierarchical organization on both memories, strong correspondence between long-term and short-term structures, and cumulative learning of skill hierarchies. We support claims for* ICARUS*' generality by reporting our experience with driving and three other domains. In closing, we discuss limitations of the current architecture and propose extensions that would remedy them.*

1. Introduction and Motivation

Research on intelligent agents cannot progress beyond a certain level without addressing issues of system integration. Developing improved component mechanisms has its place, but it cannot lead to full understanding unless we explore ways these components interact to produce intelligent behavior. System integration requires some commitment to an *architecture* that specifies key modules and interactions among them. Some architectural frameworks, such as those for multi-agent systems, indicate communication protocols among components but make few other commitments. In contrast, a *cognitive architecture* (Newell, 1990) places constraints on the components themselves that embody theoretical claims about the nature of intelligence. Our research on intelligent agents falls within this paradigm.

A cognitive architecture specifies the infrastructure for an intelligent system that remains constant across different domains and knowledge bases. This infrastructure includes a commitment to formalisms for representing knowledge, memories for storing the domain content, performance processes that utilize this knowledge, and learning mechanisms that acquire it. Research on cognitive architectures aims to support the same broad capabilities as humans exhibit, and usually attempts to remain at least qualitatively consistent with established psychological findings. Moreover, a cognitive architecture typically comes with some programming language that reflects its theoretical assumptions and that one can use to construct intelligent systems.

In this paper we describe ICARUS, a cognitive architecture that builds on previous work in this area but that introduces some novel features. The best method for evaluating an architecture remains an open question, but it seems clear that this should happen at the systems level rather than in terms of isolated phenomena. We will not claim that ICARUS is superior to other candidates along any particular dimension, but we will argue that it addresses facets of intelligent behavior, and the ways they fit together, that have been downplayed by other researchers on the topic.

We discuss the distinguishing features of ICARUS in the section that follows, but we should first note that most have resulted from our focus on *physical* agents. We can clarify this concern with an example domain – in-city driving – that involves cognition but in which perception and action also play central roles. In particular, consider the task of a driver who must deliver packages to indicated addresses in an unfamiliar area. The driver must attempt to achieve his multiple delivery goals, which themselves involve a combination of perception, action, and reasoning, while obeying the rules of driving and avoiding collisions with other cars.

To support our research on such complex tasks, we have implemented a simulated environment for in-city driving that simplifies many aspects but remains rich and challenging. Objects take the form of rectangular parallelepipeds that sit on a Euclidean plane. These 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. The physics for collisions is simplified, with vehicles exchanging momentum along their lengthwise axes.

Most vehicles are drones controlled by the simulator, but one vehicle is driven by an ICARUS agent, which has access to the same effectors and can only sense objects closer than 60 feet. The system perceives other vehicles (with no occlusion) and buildings, both described in agent-centered polar coordinates that give the object's distance, angle, relative velocity, and orientation. The ICARUS agent also perceives its distance and angle with respect to lane lines, and some of its own properties, like speed and steering wheel angle. To support the delivery task, the agent can perceive the street, address, and cross street for each package it carries, along with the current street name, the upcoming cross street, and the address associated with visible buildings. We also provide the system with top-level goals to deliver these packages to their destinations.

Despite the idealized nature of this environment, it forces us to take seriously the goal of integrating cognition with perception and action in ways that provide the same breadth and richness observed in human behavior. Thus, we will use this task domain as a running example throughout our discussion of ICARUS' features in the next section. However, generality is an important criterion for a successful agent architecture, so we follow this with a summary of results both on driving and on three quite different domains. We conclude with comments on ICARUS' relation to other frameworks and some proposals for future research.

2. Distinctive Characteristics of ICARUS

Our framework shares many features with previous cognitive architectures, such as Soar (Laird et al., 1987), ACT-R (Anderson, 1993), and PRODIGY (Minton, 1990). These include a commitment to symbolic representation of knowledge, utilization of pattern matching to select relevant knowledge elements, organization of performance into a recognize-act cycle, and an incremental approach to learning. However, ICARUS also has some distinctive characteristics, which we contrast here with the assumptions in these more established frameworks.

2.1. Primacy of Perception and Action

Most cognitive architectures draw heavily on results from the study of human problem solving. This influence is perhaps most apparent in Soar, which incorporates Newell's (1980) *problem space* hypothesis. This states that all cognitive behavior can be cast as search through a problem space which involves selection of operators to apply and states to expand. PRODIGY makes a more specific commitment to means-ends analysis as its problem-solving mechanism. ACT-R does not take as strong a position on this issue, but most models in that framework emphasize cognitive over sensory-motor activities, following the paradigm set by Newell and Simon (1972) in their early models of human problem solving. Recent versions of ACT-R and Soar have been augmented with sensory and motor modules, whereas Kieras and Meyer's (1997) EPIC emphasizes peripheral processes, but most cognitive architectures were designed with mental processing in mind.

In contrast, ICARUS is concerned centrally with intelligent agents that exist in a physical environment. Our work to date has used only simulated worlds, but they are separate and distinct from the cognitive systems, which must perceive it through sensors and influence it through effectors. At the same time, we do not reject theories of human problem solving, as they reflect important phenomena that deserve explanation. However, we hold that problem-solving activities are not primitive but rather are built on top of, and integrated with, more primitive activities for perception and action. Indeed, our philosophy is that one should never write an ICARUS program that does not operate in some physical setting, so that the framework is relevant only for embodied agents.

Perhaps the most basic assumption of traditional theories is that cognition involves the mental inspection and manipulation of list structures. Newell and Simon (1976) later refined this into their *physical symbol system* hypothesis, which states that symbolic processing is a necessary and sufficient condition for intelligent behavior. ICARUS makes the stronger claim that mental states are always grounded in real or imagined physical states, and that problem-space operators always expand to primitive skills with executable actions. We refer to this position as the *symbolic physical system* hypothesis, which comes closer to Johnson-Laird's (1989) views on thinking with mental models.

As noted earlier, our ICARUS agent for the in-city driving environment perceives a variety of object types, each described as numeric attributes in agent-centered polar coordinates. The system can influence its situation through effectors that alter speed, turn the steering wheel, and deposit a package at the current location. Our typical runs involve a city with nine or more square blocks, with at least five buildings on each side of each block, which provides a reasonably complex environment. The resulting system exhibits much the same mixture of perception, inference, problem solving, decision making, and action that humans demonstrate when driving.

2.2. Separation of Categories from Skills

Another common feature of cognitive architectures is a commitment to representations for long-term knowledge. This often takes the form of production rules, which specify the conditions under which they will match and the actions they will carry out upon execution. Production systems have been quite successful in modeling many aspects

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

```
(parked (?self ?lane)
 :percepts  ((self ?self speed ?speed))
 :positives ((in-rightmost-lane ?self ?lane)
             (stopped ?self)))
(in-rightmost-lane (?self ?lane)
 :percepts  ((self ?self)
             (lane-line ?lane))
 :positives ((in-lane ?self ?lane))
 :negatives ((lane-to-right ?lane ?anylane)))
(in-lane (?self ?lane)
 :percepts ((self ?self segment ?sg)
            (lane-line ?lane segment ?sg dist ?d))
 :tests    ((> ?d -10) (<= ?d 0)))
```

Table 2. Nonprimitive and primitive ICARUS skills for in-city driving.

```
(driving-in-segment (?self ?sg ?lane)
 :percepts ((lane-line ?lane)
            (segment ?sg)
            (self ?self))
 :start    ((steering-wheel-straight ?self))
 :skills   ((in-lane ?self ?lane)
            (centered-in-lane ?self ?sg ?lane)
            (aligned-lane-in-seg ?self ?sg ?lane)
            (steering-wheel-straight ?self))
(steering-wheel-straight (?self)
 :percepts ((self ?self))
 :start    ((steering-wheel-not-straight ?self))
 :actions  ((*straighten))
 :effects  ((steering-wheel-straight ?self)))
```

of human cognition, but they borrow key ideas from behaviorist psychology and retain a strong action-oriented flavor, although the actions are primarily mental. Even in ACT-R, which distinguishes between a procedural rule memory and a declarative memory of facts, the latter serves primarily as a source of elements for short-term memory.

However, cognition involves more than execution of mental procedures; it also includes the recognition of *categories* and drawing of associated inferences. One can certainly model categorization using production systems (e.g., Miller & Laird, 1996), but we believe that concepts serve a different function than procedures and that they are best handled with separate representations and mechanisms. We should note that many researchers seem to agree; except for those who start from such a position, few computational models of categorization are cast as production systems.

In response, ICARUS incorporates two separate long-term stores. First, a conceptual memory contains Boolean concepts that encode its knowledge about general classes of objects and relations among them. Each concept definition includes a head, which specifies the 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.

A second long-term skill memory encodes knowledge about ways to act and achieve goals. Each skill has a head, which gives a name and arguments, and a body with a variety of fields. For primitive skills, these will 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 skill *steering-wheel-straight*, which has the effect of making the *steering-wheel-straight* concept true and is considered only when *steering-wheel-not-straight* holds.

In contrast, nonprimitive skills 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 in their heads. Table 2 shows the nonprimitive skill *driving-in-segment*, which refers to the concept *steering-wheel-straight* in its *:start* field and has four ordered subskills. The driving domain lends credibility to the architectural distinction between concepts and skills, which the ICARUS interpreter treats in quite different manners, as we will see shortly.

2.3. Long-Term / Short-Term Correspondence

An agent architecture requires more than long-term memory; it must also have short-term memories that contain dynamic beliefs and intentions. A recurring idea in cognitive science is that a short-term store should simply be the 'active' portion of some long-term memory. This relation holds for the declarative memories in ACT-R, but not for its procedural production rules, which are purely long term, and Soar does not support such a mapping in any obvious form. Theories of case-based reasoning come much closer to this theme, but these have seldom been cast as general cognitive architectures.

ICARUS enforces a strong correspondence by requiring that every short-term element be a specific instance of some long-term structure. In particular, its short-term conceptual memory contains instances of defined concepts which encode specific beliefs about the environment that the agent can infer from its perceptions.[1] For instance, this memory might contain the instance *(in-lane self g0037)*, which it can infer from the *in-lane* concept shown in Table 1. Concept instances also appear in a separate short-term goal memory, which contains literals that the agent wants to achieve. For example, *(parked self g0019)* would indicate the agent's desire to be parked in lane *g0019*. In fact, ICARUS cannot encode a goal without a corresponding long-term concept.

The architecture also incorporates a short-term skill memory, which contains instances of skills the agent intends to execute. Each of these literals specifies the skill's name and its concrete arguments, which must be known objects. For example, this memory might contain the skill instance *(driving-in-segment self g0011 g0019)*, which denotes that the driver has an explicit intention to execute the *driving-in-segment* skill with these arguments. Every short-term element must be either a concept instance (belief or goal) or a skill instance (intention), which places strong constraints on the structures ICARUS can process.

2.4. Hierarchical Structure of Memory

Another distinguishing feature of ICARUS lies in its commitment to the hierarchical nature of long-term memory. There remains little doubt that human memory has this character. Many natural categories have a componential structure, and timing studies suggest that chunk boundaries remain even in well-practiced procedures. Most cognitive architectures can model such hierarchical relations, but few raise this notion to a design principle. ACT-R comes the closest by letting productions link goals to subgoals, but the relation remains mediated by working memory elements rather than referring directly to component structures.

ICARUS provides direct support for hierarchy at the architectural level. Recall that the fields in a concept definition can refer to other concepts, and thus organize categories into a conceptual lattice, with primitive concepts at the bottom and increasingly complex concepts at higher levels. For example, the concept *parked* in Table 1 is defined using *in-rightmost-lane*, which relies on *in-lane*. Similarly, each nonprimitive skill includes a field that specifies how it decomposes into subskills, ultimately terminating in primitive skills that play the same role as STRIPS operators in AI planning systems. Moreover, skills refer to concepts in other fields, thus linking the two memories in a hierarchical manner. For example, *steering-wheel-straight* in Table 2 appears in the *:skills* field of *driving-in-segment* and refers to various concepts in its *:start* and *:effects* fields. In both memories, higher-level structures refer directly to their components by name, giving more direct indexing than in production system architectures.

[1]This correspondence does not hold for ICARUS' perceptual buffer, which stores the agent's momentary perceptions.

This hierarchical organization plays a central role in ICARUS' performance mechanisms. On each cycle, the architecture matches available concepts in a bottom-up manner to infer high-level beliefs from its immediate perceptions. Once this process has completed, skill selection occurs in a top-down manner that draws on these beliefs. The skill hierarchy defines an AND-OR tree down which the interpreter traverses on each cycle, starting from top-level intentions in short-term skill memory and terminating in executable actions. Most paths are rejected because their *:start* or *:requires* fields are unsatisfied or because their heads or *:effects* already hold.

However, many paths may remain available as alternatives. ICARUS draws on two preferences that offer a balance between reactivity and persistence. First, given a choice between two or more subskills along a path through the hierarchy, it selects the first one for which the corresponding effects are not satisfied. This bias encourages reactive control, since the agent reconsiders previously completed subskills on each cycle and, if other factors have undone their effects, executes them again. Second, given a choice between two or more applicable skill paths, the architecture selects the one that shares the most elements from the start of the path executed on the previous cycle. This bias encourages the agent to continue executing a high-level skill until it achieves the associated effects or becomes inapplicable. ICARUS' skill selection process plays the same role as conflict resolution in production systems, but factors hierarchical structure into its decisions.

2.5. Cumulative Nature of Learning

Learning has played a central role in the cognitive architecture movement, with Soar, ACT-R, and PRODIGY all giving serious attention to the issue. Each of these frameworks supports the creation of new cognitive structures, specifically production rules, based on the results of multi-step problem solving. Moreover, each of their mechanisms is incremental in the sense of learning from single experiences, which is both consistent with our knowledge of human learning and desirable for synthetic agents that operate over time. All have demonstrated transfer, either across problems or within individual tasks, that produces more effective cognitive behavior than without learning. However, the mechanisms they propose offer no explanation for the origin of hierarchical skills or the cumulative nature of human learning, which builds more complex structures on top of those acquired earlier.

ICARUS' approach to learning shares important features with other architectures, such as being intertwined with problem solving and driven by impasses. When the execution module cannot find an applicable skill path that is relevant to the current goal, it invokes means-ends analysis to resolve the impasse. This involves backward chaining off skills that would achieve the goal or, if none are known, off literals in the goal concept's definition. Such choices are pushed onto a goal stack that specifies subgoals and skills that will achieve them, which are then executed if applicable or which produce further chaining if not. If the problem solver reaches a dead end or the stack grows too deep, it backtracks and tries other options, carrying out a depth-first search through the problem space.

Because means-ends analysis decomposes problems into subproblems and because the problem solver can utilize knowledge produced from previous tasks, skills that ICARUS learns for lower-level tasks are available for incorporation into skills it acquires for higher-level ones. This ability supports cumulative learning in that new structures can refer directly to those acquired from earlier experience. Nor does the architecture require sophisticated analysis to determine skills' contents. Rather, the head of a new skill is just the subgoal that led to the impasse and its subskills simply refer to the ordered subgoals achieved in its solution. If the solution involved chaining off a skill, the new structure's start conditions are the conditions of the first subskill; if it involved chaining off a concept, they are the subconcepts satisfied at the outset. This simple scheme lets ICARUS learn hierarchical and even recursive skills from the traces of successful problem solving in a cumulative manner.

3. Experiences with the Architecture

We believe that our design for ICARUS is internally consistent and well suited for physical agents, but whether it supports embodied intelligent behavior is an empirical question. To obtain evidence to this effect, we have used the architecture to develop agents for a number of domains, four of which we discuss here. Our aim has not been to produce the most robust agent possible in each domain, which might be accomplished better with specialized programs, but to demonstrate ICARUS' general ability to support reasonable behavior across a range of environments. We have reported details of these studies elsewhere (Choi et al., 2004; Choi & Langley, in press; Langley et al., 2004).

3.1. In-City Driving

We described package delivery task earlier, but we should we report our experience with the ICARUS agent we have developed for it. The system 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 model 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 run the program on different instances of the package delivery task, most involving a city with nine square blocks, a few other vehicles, and multiple packages with different target addresses. The agent reliably drives within a lane, slows for intersections, gets into the proper lane before making turns, and completes them successfully. The system slows to avoid collision when it comes behind a slower vehicle. If the agent comes upon the cross street marked on a package, it turns and continues until it can turn onto the target street or until it reaches the end, when it makes a U turn. Once on the target street, the agent continues if numbers are changing in the right direction or makes a U turn otherwise. Upon reaching a package's address, it deposits the item and drives on so it can deliver the others.

We have also demonstrated cumulative learning on a subset of these driving skills. In these runs, we provided the architecture with eight primitive skills for changing the vehicle's speed and wheel angle, along with 19 relevant concepts. We presented the system with the task of driving straight in a lane, which requires altering the wheel angle. In response, the architecture invokes means-ends analysis to construct a solution, which it executes and stores as a new skill. We then posed the more complex task of changing lanes, which again requires problem solving. However, the agent utilizes its previously learned skill to simplify the task, leading it to create another skill that incorporates it as a subskill. Finally, we asked the agent to park in the rightmost lane, which involved further problem solving and learning. When given analogous tasks later, the agent simply retrieves and executes these skills without need for problem solving.

3.2. Multi-Column Subtraction

A routine but complex task that has received attention by cognitive scientists is multi-column subtraction. This domain is less dynamic than in-city driving but represents another important class of abilities that an intelligent system should demonstrate. To reproduce behavior in this domain, we developed an environment in which the perceivable objects correspond to digits that have an x position, y position, a numeric value, and a status (clear or crossed out). Executable actions include writing down a new digit, crossing out a digit, and replacing one value with another.

We have manually developed an ICARUS program that includes concepts for grouping digits into columns, recognizing row adjacencies, and noting when columns have been processed. There are primitive skills for taking a difference, adding ten, and decrementing by one, along with two hierarchical skills, one for the top-level task and a second for borrowing. The latter has one clause for simple borrowing and another for borrowing across zero, which invokes itself recursively. The system has a similar flavor to VanLehn's (1990) treatment of multi-columnn subtraction, which also has a hierarchical organization.

The ICARUS program solves arbitrary subtraction problems in the standard manner. The agent calculates answers to the columns in a right-to-left fashion, borrowing from the adjacent or more distant columns as necessary, and writes the answers it obtains in the third row. We have not yet demonstrated learning on this domain because the goal requires that one enter a digit for each column; this involves universal quantification, which the problem solver and learning mechanism do not yet support. However, we intend to extend both modules along these lines in future versions, which should let them acquire hierarchical skills for this task.

3.3. The Blocks World

The Blocks World is a classical AI planning domain that involves a table, a set of blocks, and a gripper. Any number of blocks may sit on the table, but only one block may sit on another. Actions include lifting a block from the table or another block and placing a block on the table or another block. For this domain, we provided ICARUS with nine concepts and four primitive skills, along with one concept for each of four distinct goals. Unlike multi-column subtraction, the Blocks World allows not only different initial states but also distinct goal configurations.

We utilized this domain primarily to evaluate ICARUS' modules for problem solving and learning. The primitive skills are sufficient, given enough computational resources, to solve any task, but means-ends analysis requires extensive effort when there are more than a few blocks. Thus, we presented the system with ten problems each with three, four, five, and six blocks, which are easy enough for the problem solver to handle. The learning mechanism produces 18 new skills from the solution traces, including some with recursive calls. We then turned learning off and tested the agent on 30-block problems, nearly all of which it handles with little effort. In this domain, ICARUS' learning mechanism provides excellent generalization to tasks with many more objects than ones on which it is trained.

3.4. FreeCell Solitaire

FreeCell is a solitaire game that involves stacks of cards on eight columns, all faced up and visible to the player. Four free cells are available as temporary holding spots for one card at a time, and four foundation cells correspond to four different suits. The goal is to move all cards from the eight columns to the foundation cells in ascending order and grouped by suit. Only cards that are positioned on top of each column's stack and those in free cells can be moved; these can be shifted to a free cell, to the proper foundation column, or to an empty column. We again devised a simulator that lets the agent make legal moves and perceive the status of cards and cells.

For this domain, we did not provide the agent with any high-level skills, but we gave it 24 concepts and 12 primitive skills that are sufficient, in principle, to handle any solvable configurations. However, with this knowledge the agent could handle only simple problems that involved a few cards or started near the goal state, so we again trained it in a cumulative manner. For example, after being trained on FreeCell problems with 8 and 12 cards, the learned recursive skills are sufficient to handle most 16 and 20 card tasks using only reactive control. Also, because ICARUS allocates limited resources to problem solving, training on simpler tasks greatly increases the chance of success on more complex problems.

4. Related and Future Work

We argued earlier that ICARUS incorporates a number of features that distinguish it from traditional cognitive architectures. However, this does not mean related ideas have not appeared elsewhere under different guises. For instance, our approach has much in common with the 'reactive planning' movement, which often utilizes hierarchical procedures that combine cognition, perception, and action in physical domains. Examples include Georgeoff et al.'s (1985) PRS, Nilsson's (1994) teleoreactive framework, and Freed's (1998) APEX architecture. These lack the ability to generate novel plans by composing skills, but Albus and Meystel's (2001) RCS, Howe's (1995) PHOENIX, and Bonasso et al.'s (1997) 3T architectures combine planning with reactive control in complex environments. However, they do not transform the resulting plans into generalized reactive skills,[2] as does ICARUS.

Our framework also shares some assumptions with BDI architectures (e.g., Rao & Georgeff, 1992), which give central roles to beliefs, desires, and intentions. We can view the contents of ICARUS short-term conceptual memory as its beliefs, the agent's goals as desires, and the elements in short-term skill memory that it attaches to those goals as intentions. Some key differences are ICARUS' incorporation of ideas from cognitive science and its utilization of impasse-driven problem solving to learn new skills, both of which are associated with the cognitive architecture movement. Thus, one can view our approach as unifying theoretical concepts from these two distinct traditions.

Another close relative to ICARUS is the PRODIGY architecture (Minton, 1990), which draws on means-ends analysis to solve problems and uses an analytical method to learn either search-control rules or macro-operators from planning traces. This framework does not support reactive execution or commit to physical agents, but its approach

[2]Sun et al.'s (2001) CLARION architecture also acquires reactive control knowledge, but draws on a quite different approach that combines reinforcement learning with rule induction.

to problem solving and learning have many common features with our own. This holds especially for an extension (Veloso & Carbonell, 1993) that records problem-solving traces and solves new problems by derivational analogy with earlier ones, which does not produce generalized skills but which can exhibit effects similar to those produced by cumulative learning in ICARUS.

Despite its novel characteristics, as an agent architecture the current version of ICARUS falls short on a number of fronts. One drawback is the assumption of unlimited perceptual resources, which lets an agent sense each attribute of every object within its field of view. Clearly, humans have more limited capabilities, in that they must focus attention on an object to extract its features. We plan to treat perceptual attention as another action under the skills' control. However, this change will interact with the current assumption that conceptual inferences are removed from short-term memory when their supporting perceptions disappear. One response would be to retain inferred beliefs (e.g., that a lane is clear) across cycles but associate with them expected durations, which in turn would influence attentional decisions.

Another omission relates to ICARUS' reliance on predefined concepts to guide its skill learning. Clearly, humans can also acquire new concepts, and we should incorporate this ability in future versions of the architecture. One promising approach involves defining a conjunctive concept for the start conditions of each learned skill. This concept would be linked to the goal that the skill achieves, which suggests a different path to guiding attention and conceptual inference. Later concepts would build upon ones created earlier, producing another instance of cumulative learning. In some cases, these definitions would be recursive to handle situations that involve variable numbers of objects.

Finally, ICARUS lacks the key human ability to store its previous experiences in an episodic memory and retrieve them later. However, we have noted that our framework requires that elements in short-term memory correspond to instances of generic concepts or skills. This suggests we can model experiential memory by extending these structures to include time markers that indicate when they entered and left the short-term stores, with elements being indexed through the generic structures of which they are instances. Jones and Langley (1995) report one method for analogical retrieval that we might adapt to operate over such episodic traces.

5. Concluding Remarks

In closing, we should review the distinctive characteristics of our theoretical framework and the results to date. Unlike most cognitive architectures, ICARUS is concerned centrally with intelligent behavior in physical domains. Processes for perception and action are more basic than ones for inference and problem solving, though they interact tightly. The architecture separates concepts from skills, which are stored in distinct but connected long-term memories, and carries this dichotomy over to short-term memories, which contain instances of concepts (beliefs and goals) and skills (intentions). Both conceptual and skill memory are hierarchical, with the former directing bottom-up inference and the latter structuring top-down selection of actions. Impasses in execution lead to means-ends problem solving, whereas traces of successful solutions are stored as generalized skills in a cumulative manner.

Our experimental studies of ICARUS' behavior remain in their early stages. Nevertheless, we have shown that the architecture supports an interesting mixture of cognition, perception, and action in a variety of domains, including in-city driving, which provide encouraging evidence of generality. We have also shown that, in three of these domains, problem solving leads the system to acquire hierarchical skills in a manner that builds on previous learning. We identified some limitations of the current architecture, but these suggested in turn some natural extensions which will let ICARUS cover a still broader range of intelligent behavior that, we believe, will prove difficult to achieve in traditional architectural frameworks.

Acknowledgements

This research was funded in part by Grant HR0011-04-1-0008 from DARPA IPTO and by Grant IIS-0335353 from the U.S. National Science Foundation. Nima Asgharbeygi, Dongkyu Choi, Kirstin Cummings, Negin Nejati, Seth Rogers, and Jiang Xuan have contributed to development the ICARUS architecture.

References

Albus, J. S., & Meystel, A. M. (2001). *Engineering of mind: An introduction to the science of intelligent systems*. New York: John Wiley.

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D., & Slack, M. (1997). Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence, 9*, 237–256.

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. (in press). Learning teleoreactive logic programs from problem solving. *Proceedings of the Fifteenth International Conference on Inductive Logic Programming*. Bonn, Germany: Springer.

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.

Georgeff, M., Lansky, A., & Bessiere, P. (1985). A procedural logic. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 516–523). Los Angeles: Morgan Kaufmann.

Howe, A. E. (1995). Improving the reliability of AI planning systems by analyzing their failure recovery. *IEEE Transactions on Knowledge and Data Engineering*, *7*, 14–25.

Johnson-Laird, P. N. (1989). Mental models. In M. I. Posner (Ed.), *Foundations of cognitive science*. Cambridge, MA: MIT Press.

Jones, R., & Langley, P. (1995). Retrieval and learning in analogical problem solving. *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 466–471). Pittsburgh: Lawrence Erlbaum.

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.

Langley, P., & Cummings, K. (2004). Hierarchical skills and cognitive architectures. *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society* (pp. 779–784). Chicago, IL.

Miller, C. S. & Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science*, *20*, 499–537.

Minton, S. N. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363–391.

Newell, A. (1980). Reasoning, problem solving, and decision processes: The problem space hypothesis. In R. Nickerson (Ed.), *Attention and performance* (Vol. 8). Hillsdale, NJ: Lawrence Erlbaum.

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

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newell, A., & Simon, H. A. (1976). Computer science as empirical enquiry: Symbols and search. *Communications of the ACM*, *19*, 113–126.

Nilsson, N. (1994). Teleoreactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Rao, A. S. & Georgeff, M. P. (1992). An abstract architecture for rational agents. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (pp. 439–449). San Mateo, CA: Morgan Kaufmann.

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.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions*. Cambridge, MA: MIT Press.

Veloso, M. M., & Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, *10*, 249–278.