An Architecture for Persistent Reactive Behavior

Dongkyu Choi, **Matt Kaufman**, **Pat Langley**,
Negin Nejati, and **Daniel Shapiro**
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 integrated architecture for intelligent physical agents. The framework supports long-term memories for hierarchical concepts and skills, along with mechanisms for recognizing concepts that hold in the environment, determining which skills are applicable, and selecting for execution the skill with the highest expected value. We illustrate these processes with examples from the domain of in-city driving, and we report experimental studies on a package-delivery task that examine* ICARUS*' ability to combine reactive behavior with persistence over time. We conclude with a discussion of related work on agent architectures and our plans for extending the system.*

1. Introduction and Background

Research on agent architectures pursues a central goal of artificial intelligence: the creation and understanding of synthetic agents that support the same capabilities as humans. Such architectures aim for breadth of coverage across many domains, and they offer an account of intelligence at the systems level, rather than focusing on component methods designed for specialized tasks. They run counter to the increasing fragmentation of the field in that they provide integrated frameworks for producing complex behavior in a general, domain-independent manner.

An agent architecture – sometimes called 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 this domain content, processes that utilize the knowledge, and learning mechanisms to acquire or revise it. An agent architecture can interpret different knowledge bases, just as a computer architecture can run different programs.

In this paper, we report on the latest version of ICARUS, an agent architecture that builds on previous work in this area but also has some novel characteristics. One difference is that ICARUS includes separate memories and processes for concepts, which describe situations in the environment, and skills, which describe how to respond to them. In addition, the architecture combines the symbolic structures common to many agent architectures with the numeric value functions used in many learning systems. Finally, ICARUS supports reactive behavior but modulates it with contextual knowledge and persists in extended activities. This latter capability is a focus of the current paper.

We begin by describing a simulated driving environment that illustrates the types of domains for which we designed the architecture. After this, we examine ICARUS' long-term and short-term memories, including their formalisms for encoding knowledge, then examine its mechanisms for operating on these memory structures. Next we report experimental studies of an ICARUS agent's behavior in the driving domain, dealing mainly with the role of persistence in executing multiple high-level tasks. In closing, we consider the intellectual influences on our research and outline our plans for extending the architecture.

2. An Illustrative Domain

To support our development and evaluation of ICARUS, we have implemented a simulated environment for in-city driving. All objects in this environment take the form of rectangular parallelepipeds that sit on a Euclidean plane. These include static objects like road segments, intersections, lane lines, and buildings; they also include vehicles, the positions and orientations of which change over time. Vehicles can collide with each other and with buildings, but they roll over road-related objects without incident. Figure 1 presents a screen shot of the graphical display for a typical city.

Each vehicle is driven by an agent that can accelerate or decelerate and turn its steering wheel left or right. Associated control variables interact with realistic physical laws to determine each vehicle's motion on a given time step, so that they speed up, slow down, and change directions in reasonable ways. Collisions are handled less realistically, with vehicles simply exchanging momentum along their lengthwise axes. Most of the vehicles are drones controlled by the simulator itself. These vehicles stay in the rightmost lane and come to a near stop at all intersections. They turn at inter-

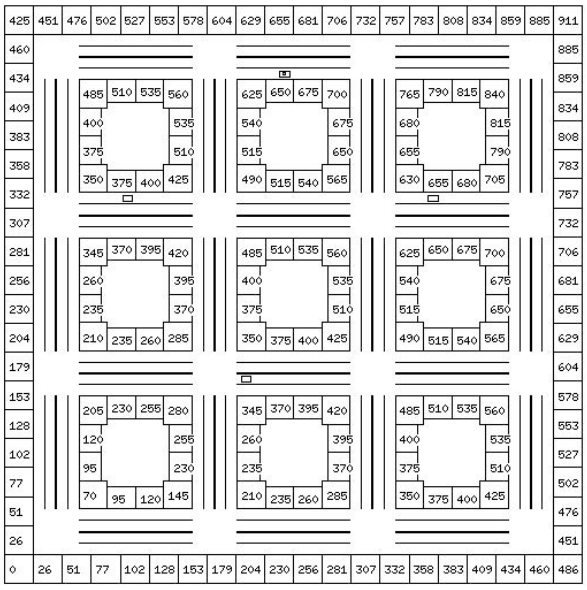

Figure 1: Display of the simulated environment for in-city driving and package delivery.

sections when these occur at city boundaries and they sometimes turn at random cross streets.

However, one vehicle in the environment is instead controlled by an ICARUS agent. It can perceive objects around it up to 60 feet away but no farther, including other vehicles (with no occlusion) and the corners of buildings, both described in agent-centered polar coordinates that give each object's distance, angle, relative velocity, and angular velocity. The ICARUS agent also perceives its distance and angle with respect to lane lines, along with its own properties, including speed and the angle of its steering wheel.

We provide the agent with top-level intentions to deliver packages to specific destinations. To support this task, it can also perceive the street, numeric address, and cross street for each undelivered package, along with the current street, the upcoming cross street, and the address associated with visible building corners. The ICARUS agent does not have a map of the city, so it must drive around in search of the target addresses, stopping to unload the appropriate package whenever it finds one. Of course, it must drive safely in the process, staying on the right side of the road, making necessary turns, and avoiding collisions along the way. Taken together, these constraints produce a challenging task environment that requires integration of perception, reasoning, and action, as well as a combination of agent reactivity and persistence.

3. Memories and Representations

An integrated architecture should make some commitment to its representation of knowledge and the memories in which that knowledge resides. In this section we describe ICARUS' memories for long-term knowledge and short-term beliefs, along with the general forms taken by the contents stored in them.

3.1 Long-Term Conceptual Memory

ICARUS incorporates a long-term memory for Boolean concepts that encodes its knowledge of familiar situations. This can include descriptions of categories for isolated objects, like types of vehicles, but also physical relations among objects, such as the relative position of two vehicles or buildings. These concepts correspond to the traditional notion of logical categories and provide ICARUS' vocabulary for describing its experiences. Each entry specifies the concept's name and its arguments, along with the optional fields `:percepts` (which describes perceptual entities that must be present), `:positives` (which gives lower-level concepts it must match), `:negatives` (which states lower-level concepts it must not match), and `:tests` (which specifies numeric relations it must satisfy).[1]

Table 1 presents some concepts from the in-city driving domain. For example, *corner-ahead-left* describes a situation in which the ICARUS agent perceives a corner with an angle (measured in agent-centered radians) in its forward left quadrant. The concept *in-intersection* matches when the agent perceives a *near-block-corner* that resides behind it, has labeled another as a *corner-straight-ahead*, and has not noted any *far-block-corner*. The concept *in-lane* matches against situations in which the agent is on the right side of the road, it perceives a lane line to its left, and two satisfied numeric tests ensure it is centered in the lane.

Taken together, these definitions organize ICARUS categories into a conceptual hierarchy. Primitive concepts are defined entirely in terms of perceptual conditions and numeric tests, but higher-level concepts can also incorporate other concepts in their definitions. The actual form is a lattice, with primitive concepts occurring at the bottom, concepts defined in terms of them immediately above, and more complex concepts at higher levels. Structurally, this lattice bears a close resemblance to the Rete networks (Forgy, 1982) used for matching in production-system architectures.

3.2 Long-Term Skill Memory

To complement its conceptual memory, ICARUS incorporates a long-term skill memory that encodes knowledge about ways to act and achieve goals. This contains specifications for skills that apply in certain situations and that produce desired effects. Skills provide ICARUS with a repertoire of behaviors that let it influence the physical situations in which it finds itself.

Each skill has a name, arguments, and eight optional fields. The `:effects` field specifies a conjunction of

1 Each Boolean concept also has an associated function that specifies the reward or utility the agent receives when that concept is true. However, these do not play a role in the current paper, so we will not discuss them further.

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

```
(corner-ahead-left (?corner)
 :percepts  ((corner ?corner r ?r theta ?theta))
 :tests     ((< ?theta 0)
             (>= ?theta -1.571)))
(in-intersection (?self)
 :percepts  ((corner ?ncorner theta ?theta)
             (self ?self))
 :positives ((near-block-corner ?ncorner)
             (corner-behind ?ncorner)
             (corner-straight-ahead ?scorner))
 :negatives ((far-block-corner ?fcorner))
(in-lane (?lline)
 :percepts  ((lane-line ?lline dist ?ldist))
 :positives ((on-right-side-of-road ?rline)
             (left-lane-line ?lline))
 :tests     ((> ?ldist -7)
             (< ?ldist -3))
```

known concepts that, taken together, encode the situation the skill is intended to achieve. Each skill can also include a `:start` field, again cast as a conjunction of known concepts, which specifies the situation that must hold to initiate the skill, and a `:requires` field, which must hold throughout the skill's execution. For example, Table 2 shows the skill *make-right-turn*, which has no explicitly stated effects, can start only if *?self* is at the appropriate turning distance and in the right lane (along with other conditions), and requires that *?corner* be a right block corner and that the vehicle be near enough to it.

In addition, each ICARUS skill includes other fields that specify how to decompose that skill into its subskills. An `:ordered` field indicates that the agent should consider these component skills in a particular order. For example, *make-right-turn* directs the agent to consider *enter-intersection*, *turn-past-halfway*, and *complete-right-turn*, and to select reactively from the last subskill that is applicable, since presumably this is closer to the desired effects. Alternatively, an `:unordered` field identifies a choice among subskills. For instance, the table's decomposition for *go-straight-in-lane* involves six subskills, including *bear-right-in-lane* and *slow-for-intersection*, from which the system selects, regardless of order.

A third option is the `:actions` field, in which a primitive skill like *bear-right-in-lane* specifies one or more opaque actions that are directly executable. For our driving simulator, such actions correspond to increasing or decreasing the vehicle's speed, turning its wheels to the left or right, and depositing a package. Thus, a primitive skill plays the same role as a STRIPS operator in a traditional planning system, with the `:start` field serving as the preconditions and the `:effects` field specifying the results of execution.

Actually, ICARUS specifies one or more ways to decompose each skill in this manner, much as a Prolog program can include more than one Horn clause with the same head. Different decompositions of a given skill must have the same name, number of arguments, and effects. However, they can differ in their start conditions, requirements, and subskills. For example, the skill *straighten-in-lane* has two such expansions, one for straightening the vehicle to the right and another for straightening it to the left.

Each skill decomposition also includes an expected value function that encodes the utility expected if it executes the skill with this decomposition. This function is specified in two parts: a `:percepts` field that matches against the values of observed objects' attributes and a `:value` field that provides an arithmetic function of these quantities. To date, we have restricted the latter to linear functions of the numeric descriptors matched by the skill, since this proved useful in our earlier work on reinforcement learning. For example, the expected value for *slow-for-intersection* in Table 2 depends linearly on the variables *?sd* and *?speed*, which can vary from moment to moment.

3.3 Short-Term Memories

ICARUS' long-term memories encode stable knowledge about a given domain. However, to generate behavior, the architecture also requires short-term stores that can change rapidly. These should make contact with long-term concepts and skills, but they must also represent temporary beliefs about the agent's environment and its intended activities.

One such memory is ICARUS' *perceptual buffer*, which contains descriptions of physical entities that correspond to the output of sensors. In the driving domain, this short-lived memory contains literals like *(corner c0027 r 10.53 theta 0.962 dv -0.041 dtheta 0.032)*, which describes a perceived corner named *c0027* with associated distance from the agent *r*, angle *theta*, relative speed *dv*, and angular velocity *dtheta*, all as perceived on the current time step. Other perceptual elements for the driving environment describe aspects of lane lines, the current street and upcoming cross street, the packages being carried, and the agent's own state.

In contrast, ICARUS' *short-term conceptual memory* contains instances of concepts that are defined in long-term concept memory. These literals encode specific beliefs about the environment that the agent can infer from those present in its perceptual buffer. For instance, this memory might contain the instance *(in-intersection self)*, which it can infer from the *in-intersection* concept shown in Table 1. This depends on the presence in memory of instances for the concepts *near-block-corner* and *corner-straight-ahead*, which ultimately ground out in perceptual entities.

Finally, ICARUS includes a *short-term skill memory* that contains instances of skills the agent intends to execute. Each of these literals specifies the skill's name

Table 2: Some ICARUS skills for in-city driving, including the `:percepts` and `:value` fields used to compute expected values.

```
(go-straight-in-lane (?self)
 :requires   ((on-right-side-of-road ?yline)
              (left-lane-line ?line))
 :effects    ((in-lane ?line)
              (parallel-to-road ?self))
 :unordered  ((bear-left-in-lane ?self)
              (bear-right-in-lane ?self)
              (cruise ?self)
              (speed-up ?self)
              (straighten-in-lane ?self)
              (slow-for-intersection ?self))
 :value       (20))
(slow-for-intersection (?self)
 :percepts   ((corner ?corner street-dist ?sd)
              (self ?self speed ?speed))
 :requires   ((should-slow-for-intersection ?self)
              (near-block-corner ?corner))
 :actions    ((*slow-down))
 :value      (+ (* -5 ?sd) (* 20 ?speed)))
(bear-right-in-lane (?self)
 :percepts   ((lane-line ?line dist ?d angle ?a)
              (self ?self wheel-angle ?sa))
 :requires   ((left-lane-line ?line))
 :actions    ((*turn-right))
 :value      (+ (* 2 ?d) (* 70 ?a) (* -20 ?sa) 10))
(make-right-turn (?self ?corner)
 :start     ((in-rightmost-lane ?rline)
             (right-block-corner ?corner)
             (near-block-corner ?corner)
             (at-turning-dist ?self))
 :requires  ((right-block-corner ?corner)
             (near-block-corner ?corner))
 :ordered   ((enter-intersection ?self ?corner)
             (turn-past-halfway ?self ?corner)
             (complete-right-turn ?self))
 :value      (30))
```

and its concrete arguments. For example, this memory might contain the skill instance *(make-right-turn self c0027)*, which denotes that the driver has an explicit intention to execute the *make-right-turn* skill with these arguments. In addition, each skill instance includes the expected value if executed, which is computed from the value function associated with that skill and perceptual attributes matched in its `:percepts` field. The agent uses this number to choose among skills and among alternatives within skills.

4. Interpreting and Utilizing Knowledge

Like most architectures for intelligent agents, ICARUS operates in distinct cycles. On every cycle, the system updates its perceptual buffer, determines which concepts are matched, and adds supported beliefs to conceptual short-term memory. The architecture then selects a path through the skill hierarchy and executes it, producing changes in the environment that influence decisions on the next cycle. In this section, we discuss each of these processes in turn.

4.1 Categorization and Belief Update

On each cycle, ICARUS refreshes the contents of its perceptual buffer by applying preattentive sensors to every object within a given distance of the agent. This produces a set of perceptual elements that initiate the process of matching against long-term concepts. Once these elements have been added, the matcher checks to see which primitive concepts (ones based only on perceptual descriptions) match, then adds each matched instance to conceptual short-term memory.

Recall that ICARUS organizes concepts in a lattice with primitive concepts at the bottom, concepts defined in terms of them and perceptual elements at the next level, and so forth. Once the system has determined which primitive concepts match, it checks more complex ones, at each step adding matched instances to short-term memory and then considering other concepts that depend on them. ICARUS repeats this process on each cycle, so concept instances remain in short-term memory only if they have direct support from the perceptual elements upon which they depend.[2]

4.2 Selection and Execution of Skills

As we noted earlier, ICARUS includes a short-term skill memory which contains a set of skill instances that the agent should consider executing. On each cycle, the architecture examines each such instance in detail to determine whether it applies to the current situation and, if so, which one has the highest expected value.

For each skill instance, ICARUS accesses all decompositions of the general skill and checks to see if they are applicable. A skill is applicable if, for its current variable bindings, its `:effects` field does not match, the `:requires` field matches, and, if the system has not yet started executing it, the `:start` field matches the current situation. Moreover, at least one of its subskills must also be applicable. Since this test is recursive, a skill is only applicable if there exists at least one acceptable path downward to an executable action. ICARUS considers all such acceptable paths downward through the skill hierarchy, returning the path with the highest expected value for each instance in short-term skill memory. Since variables can be bound within the body of a skill decomposition, this set may include multiple variants of each skill instance.

For each such path, the architecture computes the expected value and selects the candidate with the highest utility for execution. For a given path, it uses the value function stored with each skill and the numeric attributes matched in that skill's `:percepts` field to

2 We have also considered implementing the concept recognition process using a Rete network (Forgy, 1982) or a truth-maintenance system, but their efficiency in such dynamic environments remains an empirical question.

calculate the expected value at each level, summing the results along the path to compute the overall score. For instance, for the path *((drive self), (go-straight-in-lane self), (slow-for-intersection self)*, the system would sum the expected values for all three levels to determine the utility of slowing down. This means that the same action can have different values on a given cycle depending on which higher-level skills are invoking it, providing a way to achieve context effects.

The architecture treats a skill expansion differently depending on whether its components appear in an `:unordered` set or an `:ordered` list. If they are unordered, the module considers each of the subskills and selects the one that yields the highest scoring subpath. If they are ordered, it instead treats the list as a reactive program that considers each subskill in reverse order. If the final subskill is applicable, then it expands further only down paths that include that subskill. Otherwise, it considers the penultimate skill, the one before that, and so forth. The intuition is that the subskills are ordered because later ones are closer to the parent skill's effects, and should be preferred over earlier ones when applicable.

4.3 Reactivity and Persistence

In their naive form, reactive architectures operate as stimulus-response systems that take only the current state into account when deciding what action to execute. As we have seen, ICARUS moves beyond this simple approach by using a hierarchical organization of skills to modulate the selection of actions, but Nilsson's (1994) teleoreactive programs and Bonasso et al.'s (1997) T3 share similar capabilities. The architecture also treats ordered subskills in a special manner, but Georgeff et al.'s (1985) PRS and Freed's (1998) APEX also combine sequential constructs with reactivity.

Both approaches provide ICARUS and related systems with the ability to carry out extended activities, despite their emphasis on reactive response. However, they support such extended behavior in an all-or-none way, whereas a more flexible notion of persistence has attractions. People appear to fall on a continuum that describes how easily they are interrupted when carrying out a task or, conversely, how single-minded they are in pursuing their goals. ICARUS incorporates a global persistence parameter p that influences the agent's behavior along this dimension.

More specifically, the architecture retains a stack that encodes the instantiated path through the skill hierarchy that it selected on the previous cycle. When evaluating a candidate path with unmodified value v, ICARUS calculates the modulated path value as

$$v' = v \cdot (1 + p \cdot \sum_{i=1}^{s} k^i / \sum_{j=1}^{d} k^j) \ ,$$

where d is the depth of the candidate path, s is the number of steps it shares with the previous path, p is the persistence factor, and $0 < k < 1$ is a decay term.

For example, suppose that the agent is considering the path *((drive self), (complete-right-turn self), (straighten-wheels self)* on the current cycle, and suppose that the previous path shares the first two elements. Thus, if $p = 2$, $k = 0.5$, and the unmodified value is 10, then the modified value would be $10 \cdot (1 + 2 \cdot (0.5 + 0.25)/(0.5 + 0.25 + 0.125) = 27.14$. If the path from the previous cycle is still applicable, then the number of shared steps s equals the depth d, giving the multiplier $1 + p$.

The higher the persistence factor, the greater the agent's bias toward continuing to select the skills it picked on the previous time step. Setting the factor to zero produces fully reactive behavior that takes only the current situation into account, whereas higher values encourage the agent to repeat its previous decisions. However, such a higher setting does not rule out responses to important changes in the environment, which can cause entirely different skills or subskills to become applicable or produce large enough changes in expected values to shift behavior. An emergency situation, such as the need to slow down to avoid hitting another vehicle, can still overcome the bias toward continuing the ongoing activity, but, other things being equal, an ICARUS agent will prefer the latter course.

5. Experimental Studies of Driving

Our design for ICARUS has promising features, and we have evaluated an earlier version of the architecture on a simulated highway-driving task with encouraging results, including studies that demonstrate rapid learning (Shapiro et al., 2001). However, in-city driving is a more complex domain that is appropriate for evaluating the extended framework, which introduces separate long-term memories for concepts and skills, a short-term conceptual memory, a short-term skill memory that holds multiple intentions, and modulation of reactivity using the persistence factor. Here we report our experience with the environment described earlier.

To support basic driving in this domain, we developed an ICARUS program that includes 62 concepts (on average four levels deep) and 46 skills (on average five levels deep). The high-level skills handle issues like driving straight in a lane, getting in the rightmost lane, slowing for an intersection, driving through an intersection, turning at an intersection, and making a U turn. Informal studies revealed that these skills are sufficient to let the ICARUS agent drive in the simulated city indefinitely without serious problems. The system occasionally executes a poor turn and enters the wrong lane, but it recovers from such incidents and continues.

We also developed an extended program for package delivery that includes 19 concepts and 13 skills in addition to those for basic driving. The high-level skills here are responsible for turning on a package's cross street, turning on its target street, turning around if heading in the wrong direction, and dropping off a package at its target address. Informal studies with this

extended system revealed that it can deliver a set of packages to their specified addresses, although it does not always take the shortest route to achieve these objectives. We wrote the package-delivery concepts and skills separately from the basic driving program, then merged them with some tuning.

These results were encouraging, but we also wanted to carry out more systematic studies of the architecture's behavior. In particular, we were interested in how the persistence factor described above influences an agent's performance when it has choices among alternative activities. Our current ICARUS program for in-city driving and package delivery is mostly deterministic, with the main choices occurring at the top level with respect to which package the agent should deliver next. We hypothesized that a purely reactive agent might begin to deliver one package but be too easily distracted when it encounters streets associated with other packages. But we also hypothesized that a highly persistent agent might be so set on delivering a given package that it would not take advantage of opportunities to deliver others when they arise.

To evaluate these predictions, we require an explicit measure of the agent's performance. The natural candidate is the average time needed to deliver each package, so we calculated this statistic from a variety of runs. We created a specific city layout with four horizontal roads and four vertical roads, each four lanes wide, giving nine square blocks, as shown in Figure 1, with a total of 180 distinct addresses. We defined five separate delivery tasks, each requiring delivery of three packages from the same initial location. Our independent variable was the persistence factor, which we set to 0.0, 2.0, and 5.0. For each setting, we ran the ICARUS agent on each of the five tasks, measured the time to deliver all three packages, and averaged the results.

Figure 2 shows the results of this study, which are consistent with our expectations. The system takes longer, on average, to deliver packages when the persistence factor is either low or high than when it is has an intermediate value. Inspection of traces revealed that, in the first case, the agent tends to shift among its top-level intentions, attempting to deliver one package but shifting to another even when nearing its initial objective. In contrast, the highly persistent agent selects a package to deliver and pursues this task doggedly, even when it encounters streets relevant to other packages. The medium setting produces more balanced behavior that falls between these extremes.

We also carried out an additional study to examine the ICARUS agent's ability to scale to more complex tasks. In particular, we varied the number of blocks in the city, using the best-scoring persistence setting from the initial study. We generated cities with 9, 16, and 25 square blocks, each having the same number of horizontal streets as vertical streets. In this experiment, the average delivery time per package over four runs was 210.3 ± 60.1, 228.8 ± 81.4, and 292.3 ± 171.2, which

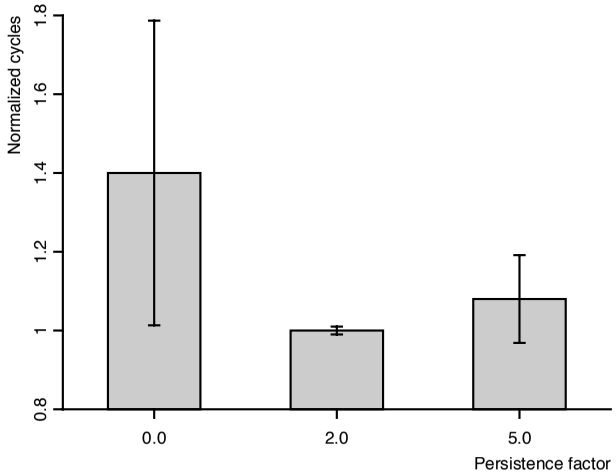

Figure 2: Average number of cycles required to deliver a package as a function of ICARUS' persistence factor, normalized by the cycles when persistence is 2.0.

suggests that the system scales reasonably as one increases the difficulty of finding the target addresses.

Of course, our experiments rely on some important assumptions that would not hold with a human driver. One is that the agent has no access to a map or directions, and must search the city until it finds the street or cross street for a package. Another is that the system does not learn routes from its driving experience, as do humans when they drive repeatedly in a city. We might encode route knowledge manually as ICARUS skills, and we plan to add such content in future versions of the driving agent. Learning such skills is currently beyond the capabilities of ICARUS, although this topic is high on our agenda. Such a facility should improve further the agent's ability to handle complex delivery tasks, but might reduce the influence of the persistence factor, which comes into play because the agent has little knowledge on which to base its decisions.

6. Intellectual Precursors

Despite its novel features, ICARUS draws on many ideas that have a long history in artificial intelligence and cognitive science. One intellectual influence comes from the cognitive architecture movement, which aims to develop integrated frameworks that support general intelligent behavior. A number of research groups have developed a variety of such architectures, two of the best known being Soar (Laird et al., 1987) and ACT-R (Anderson, 1993). Many cognitive architectures have been cast as *production systems*, which encode long-term knowledge as a set of condition-action rules that match against and modify the contents of short-term memory. ICARUS' design incorporates central ideas from this framework, including a reliance on pattern matching, but it also explicitly organizes long-term memory into concept and skill hierarchies, which differs from the implicit organization in production systems.

ICARUS also borrows from a distinct tradition of reactive control (e.g., Georgeoff et al., 1985; Nilsson, 1994; Schoppers, 1987), which emphasizes sensor-driven execution in response to changing environmental situations. Most such work takes a fully reactive approach, although some systems, like PRS, combine reactive constructs with sequential ones. More recent efforts (e.g., Bonasso et al., 1997) have combined ideas from the reactive and deliberative frameworks. ICARUS has similar goals in that it incorporates concepts from these traditions in a framework that supports physical agents that both reason and act.

Early reactive frameworks specified behavior entirely in qualitative or logical terms, but the paradigm has much in common with research on Q learning (e.g., Watkins & Dayan, 1992), which assumes stimulus-response systems that associate value functions with situation-action pairs. ICARUS extends this notion by attaching numeric functions to higher-level skills, in a spirit akin to work on hierarchical reinforcement learning (e.g., Kaelbling, 1993; Andre & Russell, 2000). A related influence comes from decision theory (Howard, 1968), which addresses value-driven decision making in uncertain circumstances. ICARUS relies centrally on the decision-theoretic notion of alternative actions that produce outcomes with different expected values.

Our general approach also has much in common with knowledge-based and case-based approaches to planning and execution. Howe (1995) and Freed (1998) describe planning systems that combine partial plans and execute them in complex environments, revising them when unexpected situations arise. Hammond (1993) even describes a program of this sort that delivers packages in a simulated driving environment. ICARUS falls more toward the reactive end of the spectrum than these systems, but the differences may lessen as we introduce planning capabilities. ICARUS also shares important ideas with Albus and Meystel's (2001) RCS architecture, which organizes knowledge hierarchically and makes a clear distinction between logical structures and value judgments.

Finally, the agent architecture we have described herein retains many ideas from earlier versions of ICARUS. Even the earliest designs (e.g., Langley et al., 1991) focused on reactive agents for physical environments, and initial versions included distinct but connected long-term memories for concepts and plans. A more recent incarnation (Shapiro & Langley, 1999) introduced reactive execution of hierarchical skills. The current ICARUS incorporates ideas from each of its predecessors, but also introduces novel features, including separate memories, both short-term and long-term, for concepts and skills, as well as the utilization of a persistence factor to influence decisions.

Every architecture for physical agents must take some position on the dual issues of reactivity and persistence. As we have noted, some commit to purely reactive control with no memory of previous decisions, whereas others augment reactive methods with sequential constructs that ensure activities happen in a specified order. To our knowledge, ICARUS is the first architecture to incorporate a flexible notion of persistence that modulates rather than overrides reactivity.

7. Directions for Future Research

Although the latest version of ICARUS constitutes a significant advance over its predecessors, the architecture still lacks many capabilities that we would expect in a general intelligent agent. One such omission relates to our framework's emphasis on execution over planning, which is important in its own right. In response, we intend to add a new module that chains backwards when the agent attempts to execute a skill with unmet requirements, along with another mechanism that supports projecting the effects of future activities on the environment. The current representation of skills should support these extensions, though we must still specify when the agent carries out such cognitive activities and how it selects among them.

Another limitation of the current architecture is its restriction to executing one skill on each time step. Future versions should support the execution of skills in parallel, but place resource constraints on this ability. This will require an expanded formalism for skills that specifies the resources they consume on each cycle. We will also need to generalize ICARUS' current method for skill selection to take expected resource consumption into account. We envision a decision-theoretic treatment that trades costs against benefits. An important special case involves perceiving the environment, which currently happens automatically through preattentive processes. A more realistic scheme would handle some perception through explicit sensing actions that require resources and thus must be invoked selectively.

Our description of ICARUS has emphasized the hierarchical nature of long-term skill memory, but, as it stands, the architecture offers no account of this hierarchy's acquisition. One promising idea involves caching the results of successful backward chaining into a higher-level skill that includes the unsatisfied skill and the repairing skill as its components. This approach is similar in spirit to methods for chunking in Soar (Laird et al., 1987) and macro-operator formation (e.g., Iba, 1989). However, previous work along these lines has constructed 'flat' knowledge elements, whereas cached ICARUS skills would retain their original structures as part of the new hierarchical skill.

Finally, like most agent architectures, ICARUS lacks any episodic memory to store its own previous experience. Knowledge about concept instances that were once true and skills that it once executed would support important abilities, such as answering questions about past events. Upon reflection, episodic memory seems closely related to short-term memory, in that it deals with specific instances of general concepts and skills. We intend to encode such memories as variants on short-term literals that include time markers to indi-

cate when they entered and left the short-term stores. Such traces will also include average statistics about the values expected and achieved when executing instantiated skills. The mechanisms responsible for retrieval from episodic memory are less clear and remain an open issue for future research.

8. Concluding Remarks

In this paper we described ICARUS, an architecture for intelligent physical agents that incorporates a number of features which distinguish it from earlier frameworks. ICARUS includes a long-term memory for concepts, which it defines as logical conjunctions of perceptual elements and other concepts, and a separate memory for skills, which it defines in terms of concepts and component skills. A categorization process deposits concept instances in short-term memory, while a separate process checks this memory to determine whether skills are applicable and utilizes numeric value functions to select among acceptable candidates.

We focused especially on ICARUS' ability to combine reactivity with persistence, which lets it respond to changes in the environment while pursuing high-level objectives. We demonstrated this ability in a simulated in-city driving domain that involved delivering multiple packages to their street addresses. Experimental studies of an ICARUS agent's behavior showed that the architecture supports this task, but also suggested that some settings for its persistence factor produced more desirable results than others.

Despite these encouraging results, ICARUS remains an immature architecture relative to older frameworks, and we outlined our plans to extend it along a number of dimensions. In general, we believe that ICARUS' value-driven approach, along with its other distinctive features, will support functionalities that are difficult to achieve in more traditional approaches. We hope to demonstrate these abilities in our future work on ICARUS agents for driving and other physical domains.

Acknowledgements

This research was funded in part by Grant IIS-0335353 from the National Science Foundation. We thank Meg Aycinena, Michael Siliski, and David Nicholas for discussions that led to many of the ideas in this paper.

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.

Andre, D., & Russell, S. J. (2000). Programmable reinforcement learning agents. *Advances in Neural Information Processing Systems*, 1019–1025.

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.

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.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*, 17–37.

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

Hammond, K. (1993). Toward a theory of agency. In S. Minton (Ed.) *Machine learning methods for planning*. San Francisco: Morgan Kaufmann.

Howard, R. A. (1968). The foundations of decision analysis. *IEEE Transactions on Systems, Science, and Cybernetics*, *4*, 211–219.

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.

Iba, G.A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285–317.

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 167–173). Amherst, MA.

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

Langley, P., McKusick, K. B., Allen, J. A., Iba, W. F., & Thompson, K. (1991). A design for the ICARUS architecture. *SIGART Bulletin*, *2*, 104–109.

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

Schoppers, M. (1987). Universal plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1039–1046). Milan, Italy: Morgan Kaufmann.

Shapiro, D., & Langley, P. (1999). Controlling physical agents through reactive logic programming. *Proceedings of the Third International Conference on Autonomous Agents* (pp. 386–387). Seattle: ACM Press.

Shapiro, D., Langley, P., & Shachter, R. (2001). Using background knowledge to speed reinforcement learning in physical agents. *Proceedings of the Fifth International Conference on Autonomous Agents* (pp. 254–261). Montreal: ACM Press.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.