To appear in *Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures*, Stanford, CA, 1991

A Design for the ICARUS Architecture

Pat Langley, Kathleen B. McKusick,* John A. Allen,*
Wayne F. Iba, and Kevin Thompson*
AI Research Branch, Mail Stop 244-17
NASA Ames Research Center
Moffett Field, CA 94035

Goals of the Research

In this paper we describe our designs for ICARUS, an integrated architecture for controlling an intelligent agent in a complex physical environment. A typical physical agent achieves its goals through *manipulation* of other objects and *navigation* between locations. For instance, an agent may pick up a cup and place it in a trash can (manipulation), or it may go out a door, down the hall, and into another room (navigation). Both are essential for survival in the physical world, and thus constitute the main tasks around which we have designed ICARUS.

The activities of manipulation and navigation in turn require the ability to recognize physical objects, places, and situations, to generate plans that achieve goals, to execute action sequences that implement plans, and to detect situations that call for a change in plans. An architecture for physical agents must include such abilities and, as we discuss in the following sections, the components of ICARUS directly support these behaviors. In addition, a robust intelligent agent should acquire new knowledge from experience and store that knowledge in memory for efficient use. Issues of learning and memory organization are also central concerns of the ICARUS effort.

Knowledge in ICARUS

The ICARUS framework uses a single underlying organization for long-term memory – a hierarchy of probabilistic concepts – to store all knowledge types. As in Fisher's (1987) COBWEB, this structure organizes concepts at different levels of abstraction, with specific cases at terminal nodes and more general concepts at internal ones. Each node in the hierarchy specifies the probability of class membership of an experience, given that the experience is a member of the parent class. Each concept node also specifies a set of attributes or roles, the values that fill each attribute or role, and the probability of each value given membership in the concept. Roles are filled by other (possibly abstract) concepts, which in turn have their own attributes or roles. However, all concepts in ICARUS are ultimately grounded in observable attributes (e.g., length is 0.37 feet) used to describe primitive objects.

One type of knowledge that ICARUS represents in this manner describes physical objects. The architecture assumes that a *composite* object (e.g., a particular person) contains a number of components (e.g., a specific head, torso, legs, and arms), which in turn can have components (e.g., a specific hand, forearm, upper arm). ICARUS represents a composite concept (e.g., the PERSON class) as a number of roles, each specifying a set of alternative components, relations among them, and their respective probabilities. Thus, one role within the PERSON concept would point to the ARM concept with high probability (say 0.999), but also (if the agent had read J. Barrie) to the HOOK concept with lower (say 0.001) probability. ICARUS represents *place* concepts in the same manner, describing them in terms of component objects and relative locations.

The architecture uses a similar approach to represent knowledge about events, plans, and actions. Because these involve change over time, the system uses *qualitative states* (Forbus, 1985) to describe them. Each qualitative state specifies a set of objects (possibly with components), an interval of time during which certain numeric attributes of these objects (such as location, velocity, or temperature) change in a fixed direction, and the derivatives of these attributes. For instance, the event of an agent picking up a cup might involve three qualitative states: (1) the arm moves down toward the cup, while the hand opens; (2) the hand closes around the cup; and (3) the arm and cup move upward, away from the cup's previous location.

ICARUS represents a *problem* in terms of an initial qualitative state, a desired qualitative state, and the differences between these states. A *plan* to solve a problem includes three additional components – an operator (which typically reduces one or more differences), an initial subplan (to transform the initial state into one meeting the operator's preconditions), and a final subplan (to transform the operator's postconditions into the desired state). Thus, ICARUS plans include more than a sequence of actions; they also

*Also affiliated with Sterling Federal Systems.

contain a problem-solving trace that produced the sequence. Operators themselves may be plans, or they may point to primitive actions (such as raising an arm) over which the agent has direct control. Abstract plans are probabilistic summaries of specific plans, containing pointers to their components – abstract states, operators, and subplans – along with associated probabilities. For example, a generic plan for picking up an object (a manipulation plan) might have three subproblems, analogous to the event described above. ICARUS uses the same approach to store route knowledge (navigation plans), with places acting as states and with operators like MOVE and TURN.

Components of ICARUS

Our designs for the ICARUS architecture call for three main components: a perceptual system (ARGUS), a planning system (DÆDALUS), and an execution system (MÆANDER). ARGUS and DÆDALUS invoke the memory system (LABYRINTH) to retrieve structured experiences from long-term memory, which include objects, states, and plans.[1] LABYRINTH first sorts each component of an experience through memory, starting at the root node of the memory hierarchy. At each level, the memory system uses an evaluation function called *category utility* (Gluck & Corter, 1985) to select the best child node in which to incorporate the experience, or to decide whether the experience is sufficiently novel to store as a separate disjunct. In the former case, the process recurs to the next level; in the latter, sorting halts. After sorting each component of an experience, LABYRINTH classifies the composite, taking into account both the classifications of the components and relations among them. The LABYRINTH module searches for the best bindings between components in the concept and those in the experience as described by Thompson and Langley (in press). If the experience has several levels in its componential structure, this recursive process continues until the highest level of the experience has been classified. For instance, to classify an object like a table, LABYRINTH would first classify its legs and top, and then use the concepts formed by these components, as well as the spatial relations among them, to classify the table instance itself. Similarly, to classify a new problem, LABYRINTH would first classify the objects in the initial and final states, then classify the two states themselves, and finally classify the overall problem description in terms of its states and the differences between them.

In order to interact with the environment, an agent must be able to perceive and recognize objects and events; this is the function of the ARGUS component. This module accepts sequences of qualitative states generated by a 'parser' subroutine, which constantly monitors objects within the agent's sensory range, detects qualitative breaks, and continually produces a qualitative description of objects and the events in which they participate. ARGUS takes these object and state descriptions, uses an attention mechanism similar to that used in Gennari's (1990) CLASSIT to focus on a subset of objects and attributes, and passes the reduced description to LABYRINTH for classification. In some cases, a call to LABYRINTH may retrieve not just an abstract state A, but also a high-priority problem for which A is the initial state. For instance, an agent might be eating supper when it notices a tiger walk into the room. In such a situation, the current task loses priority and ARGUS posts a new problem, to the active portion of memory.

The DÆDALUS component (Allen & Langley, 1990) uses a variant of means-ends analysis to generate plans. This involves breaking the initial problem (an initial state and a desired state) into subproblems on which the system recursively calls itself. The module invokes LABYRINTH to retrieve an appropriate operator or stored plan, based on the problem's preconditions, postconditions, or the differences it reduces. This generates two subproblems: one to change the initial state into another that satisfies the preconditions of the operator, and one to change the state that results from applying the operator into one that satisfies the goal conditions. The terminating case of the recursion is a problem that can be solved by a single operator whose preconditions are met. If DÆDALUS detects a loop or a dead end, it backtracks and retrieves a different operator, producing a heuristic depth-first search through the means-ends space. In cases where ICARUS has previous experience with a similar problem class, LABYRINTH will retrieve the entire plan associated with that class. In this situation, DÆDALUS carries out a form of derivational analogy, checking each subplan to determine its relevance to the problem or subplan at hand, using it as a guide if appropriate, and resorting to means-ends analysis to handle novel aspects.

ICARUS' final component, MÆANDER (Iba, 1991), is responsible for executing plans and motor schemas. In order to execute a plan generated by DÆDALUS, MÆANDER traverses the recursive plan structure, executing each step in turn. When it encounters a primitive action in the plan, MÆANDER runs the associated effectors for the specified amount of time. Barring complications, MÆANDER eventually finishes execution, bringing ICARUS to the desired state for which the plan was engineered. However, failed expectations can occur at any time during execution, which may necessitate replanning from the current (unexpected) situation. The perceptual system (ARGUS) has control over the degree of monitoring for failed expectations during execution of a plan.

[1] We have initial implementations of these components, but we have yet to enhance them in accordance with our design and combine them into an integrated architecture.

Control of ICARUS' Components

Our designs for ICARUS assume that its perceptual, planning, and execution components run asynchronously, carrying out their activities independently but storing and reading information from an active subset of long-term memory. ARGUS and DÆDALUS invoke the LABYRINTH module as a subroutine, but they operate independently of one another and of MÆANDER. The components do affect each others' behavior, but this occurs indirectly through changes to *active memory*, the portion of the concept hierarchy through which experiences have recently passed.

For example, the planner (DÆDALUS) constantly checks the active portion of memory for newly added tasks. Upon encountering a problem with a higher priority than its current focus, it interrupts processing and works on the new problem until it generates a plan or a more important task comes along. In the process, DÆDALUS calls on LABYRINTH to retrieve similar problems from memory and to store the resulting plan.

During DÆDALUS' operation, the execution module (MÆANDER) constantly inspects the developing plan, which resides in active memory. If the probability of backtracking associated with this plan is low enough, the component initiates execution as soon as there is a completed subplan. In executing a subplan, MÆANDER carries out actions that alter the environment, potentially producing new qualitative states. However, any changes in DÆDALUS' current plan, whether due to interruption or backtracking, cause MÆANDER to shift its focus as well.

The perceptual system (ARGUS) senses the resulting state and compares it to the expected state stored with the plan in active memory. The frequency of these comparisons is determined by the probability of expectation failure stored with the plan. If the predicted and observed states are similar enough, the module concludes that the operator had the anticipated effect and takes no action except to store the result in memory. However, if ARGUS detects a significant discrepancy, it adds a new problem to active memory to transform the current state into the final state specified in the original plan, giving it a priority that is high enough to interrupt work on the current problem.

ICARUS' perceptual component also continues to monitor the environment for situations that suggest entirely new problems. In some cases, classification of the current state by LABYRINTH leads to retrieval of a stored problem. In this case, ARGUS posts the new task in active memory, where the planner will notice it and consider whether to interrupt its current activity.

Characterizations of ICARUS

Now that we have examined ICARUS' memory structures, mechanisms, and overall control structure, let us consider how the framework fares on some issues that are central to integrated architectures.

Informability and Learning

Knowledge is essential for dealing with a complex physical environment, and one way ICARUS can acquire knowledge is directly from a programmer. Because object concepts, places, plans, and motor skills are all represented in a single formalism, a user can introduce new knowledge by entering an initial 'background hierarchy'. To simplify this process, one need only provide the structure of the hierarchy (i.e., nodes and the links among them) and descriptions for terminal nodes; since ICARUS assumes each node is a probabilistic summary of its children, it can easily compute descriptions for abstract nodes.

The architecture can also acquire knowledge from experience, altering its concept hierarchy to reflect perceived objects and places, generated plans, and executed action sequences. ICARUS' main learning scheme is embedded in LABYRINTH, which is responsible not only for retrieving knowledge from long-term memory but also for storing it there. As in Fisher's COBWEB, this module relies on three basic learning operations that are interleaved with the process of retrieval. First, whenever an experience is sorted through an existing node in the concept hierarchy, LABYRINTH averages its description into the probabilistic description of that node. Second, when an experience reaches a terminal node in memory, the module creates a new node N that is a probabilistic summary of the experience and the terminal node, making both children of N. Finally, if an experience is sufficiently different from all children of a node N, LABYRINTH creates a new child of N based on the experience. Note that ICARUS incorporates each experience into its hierarchy *incrementally*, modifying long-term memory in the very act of retrieval. To mitigate effects of training order, LABYRINTH can also restructure its hierarchy by merging two concepts or splitting an existing concept.

The above mechanisms are sufficient for experiences in which the attributes are directly observable, but to incorporate a composite object into a node, LABYRINTH must also determine the best *characterization* of the experience. Recall that the values of attributes in composite concepts can themselves point to other concepts, which may have similar descriptions and thus occur near each other in memory. In such cases, LABYRINTH applies an *attribute generalization* operator, which replaces these alternative 'values' with their common parent. The result is a simpler description that may implicitly include values that were not represented in the original list of alternatives.

LABYRINTH also stores and updates probabilities on three additional characteristics of plans and motor skills that can lead to changes in behavior over time. First, for each abstract problem it retains the probability that each associated operator and subplan will prove useful in solving analogous problems. For problem classes in which this probability is low, DÆDALUS will learn to stop using stored solutions, shifting from

derivational analogy to reliance on search heuristics. For problems in which the probability is high, it will instead learn to stop checking for appropriateness and use stored operators and subplans automatically, effectively planning with macro-operators. Second, for each abstract problem LABYRINTH stores the probability that a subplan or operator will be abandoned due to failure to generate a complete solution during planning. For classes of situations in which this 'backtracking probability' is low, MÆANDER will learn to begin executing initial parts of the plan without waiting for a complete solution. Finally, for each state in an abstract plan, LABYRINTH retains the probability that this state (or one matching it) will actually result if the plan is executed. Although ICARUS operates in closed-loop mode by default, for classes of plans that have a low probability of violated expectations, the agent will learn to shift toward open-loop processing.

Generality

As noted earlier, we have designed ICARUS to control a physical agent handling tasks like object and place recognition, navigation, and manipulation. However, the architecture is general enough to suggest applications in other areas of artificial intelligence. These include diagnosis of devices and diseases, problem solving in abstract domains such as mathematics, and even natural language processing. In each of these domains, ICARUS should be able to represent and organize knowledge, retrieve and use it efficiently, and acquire it from experience. Our long-term aim is an architecture as general as that embodied by the human information-processing system, but our research bias is to focus on sensori-motor tasks (involving some planning components) and to explain abstract behavior as an outgrowth of these basic operations.

Versatility

Taken together, ICARUS' components interact to support a broad range of behaviors and methods. As mentioned above, the architecture retains statistics about the probability of successful plan reuse, which lets it mimic planning with derivational analogy, search control rules, and macro-operators. In a similar manner, the storage of backtracking probabilities lets ICARUS vary the degree to which it interleaves planning and execution, whereas information about violated expectations supports the continuum from closed-loop to open-loop behavior. In addition, the notion of goal priorities, combined with ICARUS' interruption mechanism, allows both single-minded and distractable planning. Also, DÆDALUS takes preconditions into account during operator retrieval, giving aspects of both forward chaining and means-ends analysis. Moreover, the planner takes advantage of abstractions when they are available, falls back on specific cases when necessary, and resorts to systematic search when neither is present in memory.

However, ICARUS is not as versatile as one might desire. For instance, DÆDALUS only supports the generation of totally ordered plans, and it cannot produce abstract or conditional plans. Although qualitative states with duration can be used to represent both achievement goals and maintenance goals, the planning component can handle only the former, and our design for ICARUS lacks even a representation for avoidance goals. Another basic limitation involves the structure of long-term memory, which assumes that each concept has a single parent in the 'is-a' hierarchy; thus, a given experience can be classified in only one way. These limitations may be removed in future versions of ICARUS, but they are not handled by the current design.

Rationality

A fully 'rational' agent would bring all knowledge in memory and in its environment to bear in perception, planning, and execution, but time constraints force ICARUS to manage with heuristic approaches. For instance, rather than attempting to process all sensory input, the architecture uses its attention mechanism to focus on features that appear relevant for prediction. Gennari (1990) reports experiments that suggest attention increases recognition efficiency with little degradation of predictive accuracy. The transition from closed-loop to open-loop behavior carries similar advantages, even though automatized skills occasionally lead to accidents and errors. ICARUS also satisfices with respect to planning, employing a limited search that is directed by stored heuristic knowledge. In some cases this approach leads to nonoptimal plans or even to planning failures when solutions exist, but generally it produces useful plans with reasonable effort. Even retrieval is heuristic in nature, using a greedy approach that sometimes 'forgets' experiences stored in memory, but that is generally both efficient and accurate.

Taskability

ICARUS supports the ability to switch readily from one task to another. Recall that the architecture describes tasks or problems in terms of a current state and a desired state; thus, initiating a new task requires the creation of a new state pair with high enough priority to override the ongoing problem. An external user can propose such tasks via explicit commands, which typically would be described by reference to states and actions that already exist in the agent's long-term memory such as CLEAN THE CARPET, FETCH THE PAPER, and DANCE A JIG. When the user specifies a command, ICARUS retrieves a goal state and passes it along with the current state to the planner. A more sophisticated approach would allow the user a more open representation language that can describe novel desired states (e.g., RIDE A UNICORN) or actions (e.g., DANCE THE TWIST) in terms of sensori-motor primitives.

In addition to receiving outside commands, the agent can also respond to internal *drives*. These are

structures in memory – stored problems – that match against key situations, such as noticeable hunger or extreme fatigue. Classification of a new state by the perceptual system can lead ICARUS to retrieve a desired state that differs from the one currently being pursued. If the retrieved goal has higher priority than the current one, the planner will interrupt its activities and switch to the new task. Once the problem has been solved, control will pass back to the original task, unless another one has taken over in the meantime.

Reactivity

ICARUS may face environments that range from stable to changing and predictable to unpredictable, and this continuum of situations calls for varying degrees of reactivity. As we have mentioned, the architecture can adjust the degree to which it monitors the environment while carrying out actions, thus supporting behaviors that are appropriate to these different situations. In stable and predictable environments, the agent can monitor infrequently and thus execute actions more automatically, freeing attentional resources for other purposes. In contrast, an unstable and unpredictable environment would require frequent monitoring. For example, if the agent is strolling on a small desert island on a calm day, it can devote little attention to its path and appreciate the waves and clouds. However, on a windy day it must be on constant guard for falling coconuts, and so must attend carefully to its path and nearby palms. Reactivity is not always necessary or desirable, and ICARUS can behave appropriately for its current environs.

Another aspect of reactivity involves interruption, and earlier we discussed ARGUS' facility for posting new problems that can interrupt ongoing plan generation. However, such interruption is undesirable if the agent is working on a high-priority task and the newly proposed problem is trifling. Under these circumstances the agent should ignore distractions in the environment and operate in a single-minded manner, like a traditional planner. On the other hand, if the current problem is less urgent, it should be easier for a new task to interrupt planning and redirect the agent's attention. ICARUS handles these different situations with priorities, which produce a range of behaviors from reactive to single-minded.

Scalability and Efficiency

Our designs for ICARUS include the ability to bring diverse experience to bear on larger problems that the agent has never before encountered. For example, in navigation, the agent could use pieces of routes as 'macro-operators' to handle to more complex route-planning tasks. To the extent that known components can be used, and that acquired knowledge can limit search, ICARUS should have no problem scaling to larger planning tasks. ICARUS' ability to perceive the environment should also scale up well. Knowledge of relevant objects and attributes should let the attention mechanism deal with quite complex sensory input, including that resulting from motor skills involving many joints. In addition, parsing the sensor stream into qualitative states should greatly reduce the processing needed to handle complex actions and events.

Although the hierarchical organization of memory, combined with attention, serves to minimize the costs of retrieval, ICARUS does have the potential for 'expensive chunks'. In particular, when LABYRINTH sorts an experience through memory, it must compute a plausible partial match against each concept along the path. The current implementation uses a greedy algorithm that examines the conditional probabilities of each feature. However, in some domains the abstract concepts that reside near the top of memory acquire many features and relations, each with low information content. In such cases, even a greedy algorithm becomes quite expensive, and we are exploring methods for simplifying these abstractions.

Psychological Plausibility

ICARUS has many ties to the psychological literature. In particular, the architecture uses a probabilistic representation of concepts, which is consistent with many of the psychological results reported by Smith and Medin (1981). In addition, Fisher and Langley (1990) have argued that probabilistic concept hierarchies of the kind used in ICARUS can be adapted to account for three broad empirical regularities noted in the literature: basic-level effects (Gluck & Corter, 1985), typicality effects (Rosch & Mervis, 1975), and fan effects (Anderson, 1974). The framework also has potential for modeling many phenomena in rote learning.

The architecture's approach to problem solving also has links to psychological results. Newell and Simon (1972) report evidence that humans appear to use means-ends analysis in novel domains, and DÆDALUS employs a variant of this process. The planning component also relies on a form of reasoning by analogy that is generally consistent with results on analogical problem solving in humans (Gick & Holyoak, 1980). Furthermore, DÆDALUS' learning mechanisms produces the *Einstellung* effect (Luchins, 1942), a well-established type of performance degradation, and in predictable domains, its ability to form macro-operators should lead to the automization of problem-solving skills (Neves & Anderson, 1981).

Finally, the MÆANDER component, which is responsible for executing actions, is also consistent with many results on human motor behavior. Schmidt (1982) reports evidence for the distinction between open-loop and closed-loop behavior, which is central to this component of ICARUS. The deterioration of performance with increased speed, predicted by ICARUS when MÆANDER's actions are less frequently monitored, is another well-documented phenomenon (e.g., Fitts & Peterson, 1964).

Historical Influences

As we have mentioned, our model assumes a uniform long-term memory that stores knowledge about objects, places, plans, actions, and events. Our approach to organizing, using, and acquiring knowledge draws heavily on Fisher's (1987) COBWEB, which serves as the underlying component of LABYRINTH. However, the historical roots of our approach go back ultimately to Feigenbaum's (1963) EPAM, which incrementally constructed a discrimination network from unsupervised instances. This work also influenced Schank's (1982) theory of dynamic memory, which claims that the long-term store is an interleaved hierarchy, that retrieval involves sorting experience through memory, and that this process leads to memory reorganization. However, our concern with psychological phenomena has led us to incorporate Gluck and Corter's (1985) notions of probabilistic representation and category utility, which they have used to explain basic-level effects.

Another central feature of ICARUS is the use of means-ends analysis to control the planning process. Newell, Shaw, and Simon (1960) were the first to use this technique, in their GPS model of human problem solving, but very similar methods have been used in Minton et al.'s (1989) PRODIGY and Jones' (1989) EUREKA, two systems that learn in planning domains. Our approach also borrows ideas from Veloso and Carbonell's (1989) work on derivational analogy and from related research on case-based reasoning (e.g., Hammond, 1990). However, ICARUS' representation of states, operators, and problems has been influenced by work in qualitative physics (Forbus, 1985), which follows a quite different tradition.

Relation to Other Architectures

Our goal of constructing an integrated cognitive architecture has also been influenced by earlier work toward this end, particularly that concerned with learning. Some well-known examples of such architectures are Anderson's (1983) ACT*, Laird et al.'s (1986) SOAR, Minton et al.'s (1989) PRODIGY, and Mitchell et al.'s (in press) THEO. Like ICARUS, these architectures attempt to cover a broad range of behaviors within a unified theoretical framework, though they differ in their generality and theoretical content.

Each architecture clearly defines a set of memories and their characteristics, with domain knowledge residing in a long-term memory that is modified through learning. SOAR, PRODIGY, and ACT* all represent this knowledge in the form of production rules, though the latter also includes a separate declarative memory. THEO represents knowledge in frames, each containing slots that refer to other frames. The production-system architectures take no explicit stance on the organization of memory, presumably because of the assumption that production rules are matched in parallel. THEO's links between frames provide a more explicit indexing scheme, providing paths to access one piece of knowledge from many others. ICARUS also takes an explicit stance on memory organization, using abstractions to index their specializations, and using components to index the composite concepts in which they take part.

All five frameworks identify a set of primitive processes supported at the architectural level. In those based on production systems, the primitive actions involve matching condition sides and applying one or more of the matched rules. In SOAR and PRODIGY, this occurs during an elaboration cycle, in which selection, rejection, and preference rules 'vote' in favor of particular states, operators, and goals; the architecture then makes a decision based on these votes. ACT* makes less commitment about the nature of its productions, with some acting as operators, others as goal generators, and others as inference rules. In THEO the primitive operation involves slot access, which it uses to retrieve data, facts, procedures, and preferences. ICARUS diverges from these systems, using heuristic classification as its primitive operation for both perception and planning. Moreover, our architecture makes inferences (including operator preferences) through 'pattern completion' on retrieved concepts, a scheme that is quite different from the constrained directional inference rules encoded in productions and frames.

In addition, all the architectures incorporate a basic learning mechanism that constitutes a form of incremental hill climbing. Chunking in SOAR, knowledge compilation in ACT*, and explanation-based learning in PRODIGY have much in common, effectively caching the results of rule or operator applications to simplify future processing. THEO also employs a form of caching, although its stored knowledge takes the form of partial results for slot values rather than production rules. In contrast, the central learning mechanism in ICARUS is concept formation, the process of updating probabilistic descriptions and altering the structure of the concept hierarchy. This scheme is primarily inductive in nature whereas chunking and its relatives involve analytical learning.

Summary

In this paper we described our designs for ICARUS, an integrated architecture for controlling physical agents. In our framework, diverse types of knowledge are represented and organized in a unified concept hierarchy. ICARUS includes three asynchronous components for perception, planning, and execution, which call on an underlying module that updates and accesses long-term memory, and which affect each others' behavior through changes to the active portion of this memory. The framework fares well on some issues that are central to integrated architectures, and has many features that are consistent with the psychological literature.

Although we have detailed ideas about ICARUS' components, the overall architecture must still be integrated and tested. Many issues remain open, but we believe ICARUS constitutes a promising theory of intelligent behavior that deserves further exploration. It may be some time before our implementation reaches the point of controlling a physical robot, but we feel that we are rapidly making progress in that direction.

Acknowledgments

We would like to thank John Gennari and Deepak Kulkarni, who have contributed much to the design of the ICARUS architecture, and John Bresina and Steve Minton, who provided helpful comments on an earlier draft of this paper.

References

Allen, J. A., & Langley, P. (1990). Integrating memory and search in planning. *Proceedings of the 1990* DARPA *Workshop on Innovative Approaches to Planning, Scheduling, and Control* (pp. 301–312). San Diego, CA: Morgan Kaufmann.

Anderson, J. R. (1974). Retrieval of propositional information from long term memory. *Cognitive Psychology*, *6*, 451–474.

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge: Harvard University Press.

Feigenbaum, E. A. (1963). The simulation of verbal learning behavior. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought*. New York: McGraw–Hill.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Fisher, D. H., & Langley, P. (1990). The structure and formation of natural categories. In G. H. Bower (Ed.), *The psychology of learning and motivation: Advances in Research and Theory* (Vol. 26). Cambridge, MA: Academic Press.

Fitts, P. M. & Peterson, J. R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology,67*, 103-112.

Forbus, K. D. (1985). Qualitative process theory. In D. G. Bobrow (Ed.), *Qualitative reasoning about physical systems*. Cambridge, MA: MIT Press.

Gennari, J. H. (1990). *An experimental study of concept formation*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, *12*, 306–355.

Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, *14*, 385–443.

Iba, W. (1991). *A computational theory of human motor learning*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

Jones, R. (1989). *A model of retrieval in problem solving*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Luchins, A. S. (1942). Mechanization in problem solving: The effect of Einstellung. *Psychological Monographs*, *54* (248).

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

Mitchell, T. M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., & Schlimmer, J. C. (in press). THEO: A framework for self-improving systems. In K. VanLehn (Ed)., *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition*. Hillsdale, NJ: Lawrence Erlbaum.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264).

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

Rosch, E., & Mervis, C. (1975). Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology*, *7*, 573–605.

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

Schmidt, R. A. (1982). More on motor programs. In J. A. S. Kelso (Ed.), *Human motor behavior: An introduction*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Smith, E., & Medin, D. (1981). *Categories and concepts*. Cambridge, MA: Harvard University Press.

Thompson, K. & Langley, P. (in press). Concept formation in structured domains. In D. H. Fisher & M. Pazzani (Eds.) *Computational approaches to concept formation*. San Mateo, CA: Morgan Kaufmann.

Veloso, M. M., & Carbonell, J. G. (1989). Learning analogies by analogy – the closed loop of memory organization and problem solving. *Proceedings of the DARPA Workshop on Case-based Reasoning* (pp. 153–158). Pensacola Beach, FL: Morgan Kaufmann.