Orca: Intelligent Adaptive Reasoning for Autonomous Underwater Vehicle Control*

Roy M. Turner
Marine Systems Engineering Laboratory
Marine Science Center
Northeastern University
Adams Point
Nahant, MA 01908
E-mail: rmt@unh.edu

Abstract

Real-world problems demand adaptive problem solvers that can tailor their behavior to their task domain, both during a problem-solving session and over time across sessions. Otherwise, incomplete knowledge, uncertainty, the presence of unpredictable agents and processes, and hardware failure and imprecision will conspire to cause unanticipated events, mission failure, and possibly damage to the agent.

In this paper, we discuss Orca, a schema-based, context-sensitive adaptive problem solver. Orca is an intelligent mission controller for autonomous underwater vehicles (AUVs). Orca uses procedural schemas, which are like hierarchical plans, to control its actions. It uses contextual schemas, which are similar to generalized cases, to ensure that its behavior is tailored to its problem-solving situation. Two kinds of adaptation are discussed, short-term adaptation to meet the needs of the agent's current mission, and long-term adaptation, which changes the agent's knowledge to become better over time at performing all its missions.

*The author is grateful to the National Science Foundation for grant BCS–9211914, which supported this work in part. The author is also affiliated with the Department of Computer Science at the University of New Hampshire. Correspondence should be addressed to the author at Department of Computer Science, Kingsbury Hall, UNH, Durham, NH 03824.

Orca: Intelligent Adaptive Reasoning
for Autonomous Underwater Vehicle Control

Roy M. Turner
Marine Systems Engineering Laboratory
Marine Science Center
Northeastern University

Problem-solving systems designed to operate in the real world need to be adaptive if they are to be of any use. They must be able to handle unanticipated events and change the way they operate to meet the changing demands of their environments and task domains and to mitigate the effects of uncertainty, incomplete knowledge of the environment, and lack of domain knowledge.

Adaptation of a problem solver to its problem-solving environment can be broken into two types based on the time over which adaptation takes place (see, e.g., [10]). *Short-term adaptation* is how the agent changes its behavior to fit its problem-solving environment during the course of a single mission. For long-term missions, there may be some learning involved, but in general, other adaptive mechanisms will be needed. *Long-term adaptation* involves learning and tailors the agent's problem-solving knowledge to better fit its overall task domain. One can think of long-term adaptation tuning over time the knowledge an agent uses for short-term adaptation.

This paper describes work on the Orca project [10; 13] addressing adaptive reasoning in a real-world domain. Orca is an intelligent, adaptive mission controller for autonomous underwater vehicles (AUVs) that is being built at the University of New Hampshire and Northeastern University. Orca is designed to be a robust controller for AUVs for ocean science missions.

Since Orca must work in a real-world domain, we have little choice but to make it adapt, both in the short and long terms, to the exigencies of its problem-solving environment. We are currently focusing on issues related to short-term adaptation, including: run-time flexibility of procedural knowledge, conditioning of behavior via "behavioral parameter" settings, and context-sensitive reasoning via the use of a priori (experiential) contextual knowledge. Ultimately, we will focus on long-term adaptation, including case-based reasoning and other learning mechanisms.

In the remainder of this paper, we discuss Orca and its domain, then look at current work on short-term adaptation. We then discuss plans for long-term adaptation, and conclude with a discussion of the project's current status and future work.

Orca and Its Domain

Although we intend Orca to be capable of controlling a wide range of real-world physical devices, our primary focus is on controlling AUVs. The AUV control domain presents challenging problems for an intelligent problem solver. First, it involves controlling a real physical vehicle, with all the uncertainty, unanticipated events, and sheer orneriness of physical devices that that entails. Second, little is known about the ocean environment; it has been said that we know less about it, Neptune's realm, than we do the surface of the planet Neptune [2]. And third, the available sensors are not very good, which adds to the uncertainty and incomplete knowledge about the agent's surroundings.

Two kinds of AUVs are or will be available to us for our work. The EAVE-III vehicles [1; 2] (see Figure 1), developed by the Marine Systems Engineering Laboratory (MSEL), are open space-frame, short-range vehicles intended for ocean engineering development and some ocean science missions. We currently have two such vehicles and are pursuing, with the UNH Cooperative Distributed Problem Solving (CDPS) research group, research targeted at using these vehicles for multiagent missions in which each would be controlled by a copy of Orca.

The second vehicle is a long-range AUV (LRAUV) currently under development at MSEL (see Figure 1). This vehicle, which will be capable of long-duration, full-ocean depth science missions, will be the primary development target for Orca.

The AUVs are controlled by the MSEL EAVE software architecture [1], shown in Figure 2. This is a layered control architecture; each layer deals with more abstract information and longer planning time horizons than the layer below it. Lower levels are faster; consequently, it is to the vehicle's advantage to handle events as low in the architecture as possible. On the other hand, the higher the level, the more

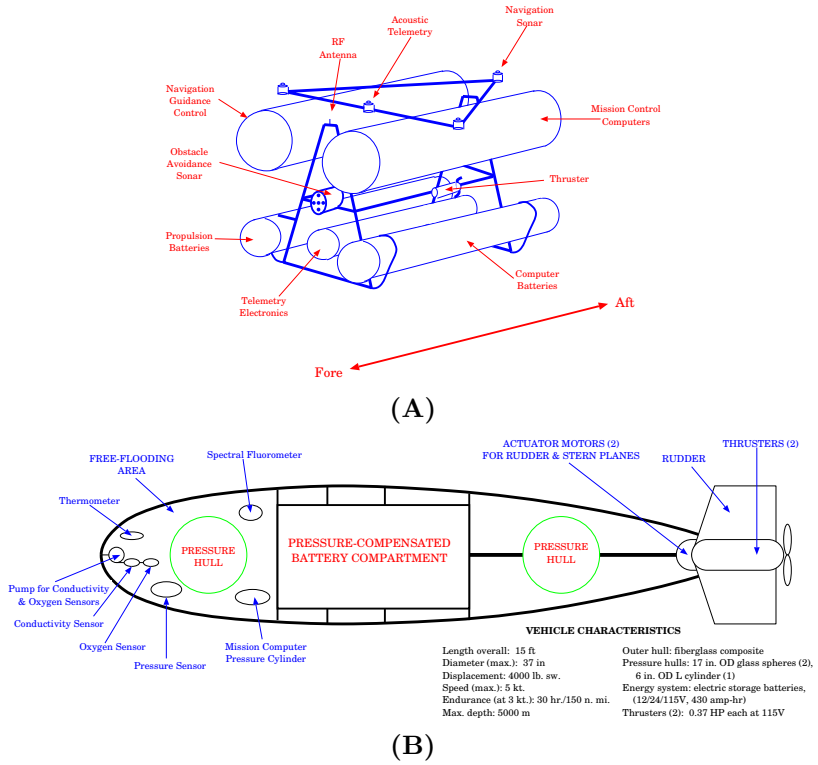

Figure 1: MSEL's AUVs: (A) EAVE-III, (B) long-range AUV.

comprehensive the knowledge, and the better responses to events can be integrated into the overall mission plan. This trade-off is handled in the EAVE architecture by allowing higher levels to set lower levels' behavioral parameters, thus conditioning their behavior. We will see below that this is one way Orca adapts its and the vehicle's behavior to the problem-solving environment. Orca will occupy the topmost layer of this architecture, which insulates it from some of the hard real-time constraints and from the second-to-second control tasks necessary to carry out missions and ensure vehicle safety.

Orca is itself composed of several distinct modules (Figure 2). All input, whether processed sensor data from the lower-level architecture (LLA), command execution status, or messages from the user(s) or other agents, arrives via the Event Handler (EH). This module updates Working Memory (WM) appropriately and decides if the new information signals an event. If so, and the event is unanticipated, EH diagnoses it, assesses its importance, and selects a candidate response [10; 11; 12] which it passes along to the Agenda Manager.

Agenda Manager (AM) is responsible for focusing attention on the appropriate goal(s) to achieve given the current problem-solving situation. When a goal is selected, Schema Applier (SA) finds an appropriate *procedural schema* [10] from Long-Term Memory (LTM), a content-addressable, conceptual memory (e.g., [5; 10]).

A *procedural schema*, or p-schema, is a specification of one way a goal or set of goals can be achieved. It is in many ways like a hierarchical plan. It is expanded at *run time* by Schema Applier into a plan network (e.g., [14]) of steps to take to achieve the goals. Each step can be a goal, a p-schema, an executable action, or a control action. Orca's p-schema language is expressive enough to represent conditional execution (if-then-else constructs), loops of various sorts, parallel and nondeterministic execution of steps, and explicit failure-handling actions and branches of the plan. When an executable action (xact) is selected for application, Orca either executes it internally (e.g., to make an inference) or issues a command to the lower-level architecture (e.g., "move to (100, 100, 20)"). SA can suspend action on a p-schema while it is waiting or when AM refocuses attention, thus allowing Orca (and the AUV) to work on more than one thing at once.

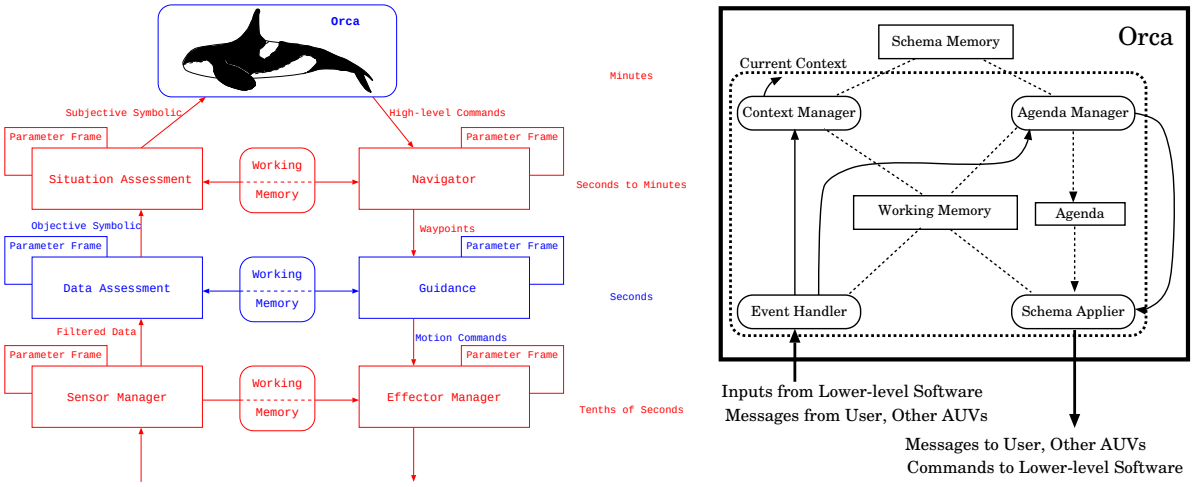

Figure 2: The EAVE software architecture (left), showing Orca's location in it, and the internal structure of Orca (right).

The Context Manager (CM) is crucial to Orca's ability to adapt its behavior to fit its problem-solving situation. It brings to bear the agent's contextual knowledge, that is, knowledge of kinds of problem-solving situations, to condition Orca's behavior appropriately to the current situation. CM impacts all facets of Orca's behavior. The contextual knowledge is represented as *contextual schemas* [9; 10], which are similar to but more general than the cases used in case-based reasoning. C-schemas contain experiential knowledge and can come either from humans or from the agent's own problem-solving experiences.

Short-term Adaptation in Orca

Short-term adaptation has to do with changes in a problem solver's behavior during the course of a mission. In many "traditional" AI planning systems, any short-term adaptation was done by the execution monitor. A ground swell of dissatisfaction with this approach began a decade or more ago (e.g., [7]) and resulted in problem solvers that were capable of changing at run-time the way solutions are created, not just the way the solutions are carried out.

Orca is positioned firmly within this approach to problem solving. Orca plans, but does not plan to the level of complete detail before beginning execution. Orca decides how to achieve goals and subgoals, and how to carry out pre-compiled action patterns (p-schemas), based on its evolving problem-solving situation. This is facilitated not only by the representation of p-schemas, but also by their organization in its memory. Orca also conditions its own and the LLA's behavior via "standing orders," that is, behavioral parameters that are set and goals that are automatically activated/deactivated based on the situation. During execution, Orca also remains ready to respond to unanticipated events that may arise.

All of Orca's behavior is affected by its context, as mediated by its Context Manager. This is another kind of adaptation, in this case an automatic adaptation to the current situation based on a priori knowledge. Case-based reasoning can also be used in much the same spirit.

In the remainder of this section, we discuss each of these facets of Orca's short-term adaptation ability. In the next section, we discuss longer-term adaptation.

Flexibility of procedural schema application. The simplest form of adaptive ability in Orca comes from the way Schema Applier applies procedural schemas. Given the expressiveness of the p-schema representation language, a single p-schema can generate a fairly wide range of actual action patterns as a result of conditional execution, loops, and so forth. In addition, a future version will incorporate a constraint "language" for describing objects in the world, which should further increase p-schemas' flexibility.

Steps in p-schemas can specify particular executable actions to take, goals to achieve, or other p-schemas to apply. Both of the latter two kinds of steps also aid Orca's ability to adapt to its run-time environment. As in several other planners (e.g., [14]), a step specifying a goal is a kind of delayed commitment that allows the planner to put off until the last moment selecting actions to take. Unlike other planners, however, a step specifying a p-schema (i.e., a sub-plan) does something of the same thing. When SA encounters such a step, it tries to specialize the p-schema (see below) to one that better fits the current situation. The p-schema specified is taken to be a suggested starting point for a search for what to do.

Adaptation via the procedural schema repertoire. A large measure of Orca's ability to adapt to its problem-solving situation is due to the content and organization of its p-schema repertoire. Orca will have a wide variety of p-schemas, some very specific for achieving particular goals in particular situations, others much more general. For example, there may be schemas for the goals "perform a search in Portsmouth Harbor" (i.e., achieve the state of Portsmouth Harbor having been searched for some object or objects), "perform a search in a harbor," "perform a search in shallow water," "perform a search," and "achieve a state."

When Orca focuses its attention on a goal to achieve, it searches its long-term (schema) memory for an appropriate p-schema to use.[1] Which schema is found is a property of the problem-solving environment and the program's memory. If it has a highly specific schema for the goal, Orca will find and use it; this will give the best anticipated result. If not, however, it will automatically use the best schema it can find. In the worst case, it will find only very general-purpose p-schemas, such as "achieve a state," which, as in MEDIC [10], will then cause it to engage in "from-scratch" reasoning, such as case-based reasoning, plan generation via nonlinear planning, or other mechanisms.[2] The end result is a homogeneous problem-solving mechanism that makes use of specific knowledge when it has it and falls back on general reasoning mechanisms if necessary. This will go a long way toward bridging the gap between reasoning mechanisms that are expert, but brittle, and those that are general-purpose, but sub-optimal.

As mentioned above, Orca tries to specialize p-schemas during execution as well. When a p-schema is encountered as a step of another p-schema, the Schema Applier tries to specialize it to better fit the current situation before expanding and applying it.

Behavioral parameters and standing orders. A real-world system requires the ability to automatically change the *character* of its activities based on the situation and the ability to manage its behavior with an eye toward homeostasis. We call the parameter settings, constraints, and goal activation/deactivation that does this the agent's "standing orders" for a situation.

Some facets of implementing standing orders for a given situation interact with problem solving in general. Problem solving may be used to generate or effect standing orders, and standing orders may constrain the way problem-solving goals are achieved or actions carried out. For example, one reasonable standing order for an AUV in the context of being in a harbor is to tighten its depth envelope, both to avoid bottom clutter and surface traffic. Alternatively, standing orders may generate goals for problem solving, or may set behavioral parameters for the agent or its underlying software and hardware that interacts only indirectly with problem solving. For example, in the context of docking, it makes sense to turn off obstacle avoidance behavior, which may be accomplished by setting a parameter in the AUV's low-level architecture.

In our approach, standing orders come primarily from contextual information, as discussed below. When entering a context, Orca's Context Manager (CM) automatically activates or deactivates goals and sets behavioral parameters based on what predictions it can make about what constitutes appropriate behavior in the situation. The result is a smooth and automatic adaptation of the AUV's behavior to the problem solving situation.

Unanticipated event handling. Event handling is one of the key abilities needed by an adaptive reasoner that is to operate in an uncertain, possibly hostile world. Too often in the AI literature, however, unanticipated events are overlooked. When attention *is* paid to them, most of the time it is with an eye

[1] The current c-schema may provide a starting point for the search; see below.

[2] General-purpose problem-solving schemas are not implemented in this version of Orca.

toward failure handling only, though they often occur well in advance of the failure they predict. Instead of waiting for a failure, events should be handled at once when they occur.

Unanticipated events occur due to uncertainty, incomplete knowledge, and the presence of unknown or unpredictable processes and agents in the environment. They are difficult to detect, difficult to diagnose, and difficult to respond to appropriately. Yet if they are not handled, they can severely impact the mission or even the likelihood of survival of the agent. On the other hand, some unanticipated events are beneficial, and these, too, should be responded to in order to seize opportunities that might otherwise pass by.

In Orca, the Event Handler (EH) is the module responsible for handling all events, anticipated and unanticipated alike. EH receives all input to Orca, both that from the low-level architecture and messages from other agents and the user(s). It watches the "input stream" and looks for patterns it can recognize as events.

Orca's event-handling process is currently in transition from a simple, linear version that uses a fuzzy rule-based system to one less linear that borrows ideas from AI in medicine's abductive reasoning research [3]. Both versions are discussed elsewhere [11; 12] and are shown in Figure 3. Both involve detection, diagnosis, importance assessment, and response selection, and both make heavy use of contextual knowledge provided by CM. The newer version will have the ability to make provisional event-handling decisions, to interact with SA, and to do more sophisticated diagnostic reasoning than the current version.

One can think of standing orders and event handling operating as two halves of one adaptive process. Standing orders condition the agent's behavior to fit the situation when all is going well, and event handling does the same thing in exceptional situations.

Context-sensitive reasoning. A major thrust of our research is in the area of context-sensitive reasoning for problem-solving systems. We have developed an approach to context-sensitive reasoning based on using a priori, explicitly-represented contextual knowledge during problem solving [9; 10]. Such knowledge is represented as *contextual schemas* (c-schemas) and organized similarly to how Orca's procedural schemas are organized. The Context Manager module watches the evolving problem solving situation and selects from the memory one or more c-schemas that represent contexts of which the current situation is an instance. These are then merged to form the *current c-schema*, a composite knowledge structure that helps control the reasoner's behavior by providing it with information specific to the current context. As the context changes, CM updates the current c-schema to reflect the new situation.

In Orca, contextual knowledge is used for several tasks, including: (1) making predictions about unseen or future states expected in the context; (2) providing a context-specific source of standing orders (behavioral parameter settings and goals to activate/deactivate); (3) context-sensitive event handling; (4) helping the Agenda Manager module maintain Orca's focus of attention appropriately; and (5) providing suggestions for procedural schemas appropriate for achieving goals in the context.

Context-sensitive reasoning of this sort is important for adaptive problem-solving behavior. It provides a way for a reasoner to adapt its behavior to fit its evolving problem-solving situation in a manner requiring relatively little problem-solving effort and that ensures that the agent's behavior is always as appropriate as possible.

In addition, this approach is a tie to long-term adaptation. Contextual schemas represent general cases of problem solving, either the agent's own or someone else's. The idea for them comes from case-based reasoning research, and in many ways they function as cases, though they are used more generally. By updating the agent's c-schema repertoire over time, the agent's short-term adaptation is itself adapted to fit long-term properties of the task domain.

Case-based reasoning. Case-based reasoning has been used for ten years or more for long-term adaptation of a problem solver's behavior (e.g., [4; 6]). Often overlooked is its potential for short-term adaptation, where "short-term" here means within the course of a single mission. If the mission is extended, then CBR based on experiences encountered earlier in the mission can help an agent solve later tasks on the same mission. We anticipate using CBR in this capacity in later phases of the Orca project, especially during long-duration missions aboard our long-range AUV.

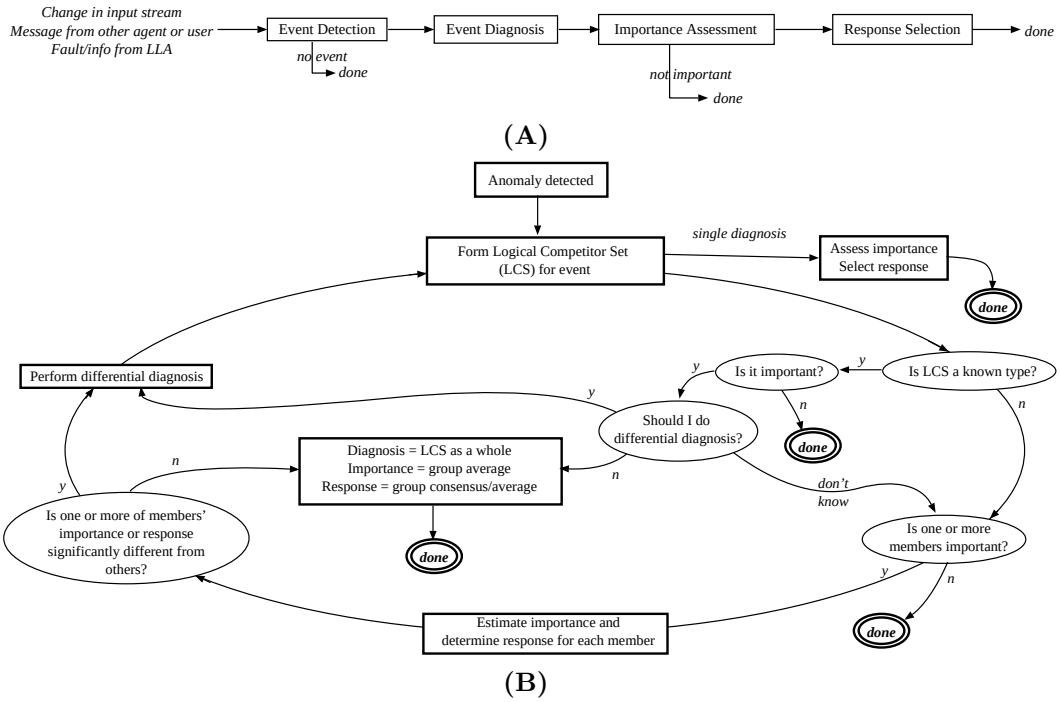

Figure 3: Current (A) and future (B) versions of Orca's event-handling process. (After [12].)

Long-term Adaptation in Orca

The other half of adaptive reasoning is long-term adaptation, or learning. We have yet to focus significant effort on this problem in Orca, though it will become an important area of research in future phases of the project. Consequently, in this section, we describe anticipated directions rather than existing capabilities or results.

Case-based reasoning. Case-based reasoning will likely play an important part in adapting Orca to its overall problem-solving domain. Orca's domain is one in which there is surprisingly little domain knowledge, a circumstance to which CBR is ideally suited. We can foresee CBR being used not only to help solve problems encountered during a mission based on similar problems encountered on past missions, but also being used to update Orca's c-schema repertoire in a manner similar to how some case-based reasoners update internal nodes in their memories (e.g., [8]). In this case, the nodes will be contextual schemas.

Contextual schema update. One way contextual schemas may be updated is as a by-product of storing cases in memory, as mentioned above and described by, e.g., Kolodner [5]. As usually implemented, this is inductive, or similarity-based, learning. This is good for those situations where there is little domain knowledge. However, when there *is* domain knowledge, it makes sense to use it. Consequently, we anticipate augmenting induction with other forms of learning, for example explanation-based learning.

Procedural schema update. We anticipate that learning procedural schemas will be much more difficult than learning c-schemas. Here the expressiveness of our representation is against us: it is difficult to see, without postulating a program that can understand general programming techniques, how p-schemas as they are currently represented can be learned—and full-fledged automatic programming is at least several years away.

We will examine this problem in future versions of Orca. It is possible that we can simplify the representation enough to allow automated learning of p-schemas without compromising too much their flexibility

and usefulness. Alternatively, we may find that it is advantageous to allow complex p-schemas to be given by humans to the program and then treated as unmodifiable, unexaminable "black boxes"; simpler schemas can then be learned by the program from its own experiences to augment the ones from the human.

Learning as schema-based reasoning. A potentially useful feature of our approach is the ability to seamlessly fold learning into the problem-solving architecture. Learning is, or can be treated as, just another kind of problem solving. Consequently, it may be possible to accomplish learning within the schema-based reasoning approach by using "learning p-schemas" and c-schemas representing being in the context of learning (as opposed to, or in addition to, doing).

By representing learning algorithms as p-schemas, we would gain the flexibility and adaptability advantages discussed above for meta-level problem solving, too. We would also open up the possibility of invoking learning not only on domain-level schemas, but also on those for learning itself, thus achieving another level of adaptability, this time in the way the problem solver learns.

Conclusion and Future Work

Two kinds of adaptive reasoning are important for real-world problem solvers, short-term adaptation and long-term adaptation. So far in the Orca project, we have concentrated mainly on the former. Orca's procedural schemas, contextual schemas, long-term memory organization, and schema application process allow it to tailor its behavior to fit its problem-solving situation with little expenditure of reasoning effort.

Future versions of Orca will begin to incorporate long-term adaptation. Long-term adaptation will be addressed by investigating ways to change the knowledge structures used for short-term adaptation. One part of this will include contextual schema update, both by induction as a by-product of storing cases and by other machine learning techniques. Another part is case-based reasoning. A third, more difficult, part will be automatic procedural schema update from experience. We anticipate learning itself being under the control of schema-based reasoning, thus allowing all of the agent's reasoning to be done by one coherent problem-solving process.

The various versions of Orca will be tested in the demanding domain of autonomous underwater vehicle control. Currently, Orca is being developed and evaluated in our SMART simulation testbed. In future phases of the project, Orca will be tested during in-water experiments aboard MSEL's AUVs.

We believe our approach is more general than this. In the future, we plan to examine its usefulness as a controller for agents involved in multiagent problem solving and for network interface agents ("softbots").

References

[1] D. R. Blidberg and S. G. Chappell. Guidance and control architecture for the EAVE vehicle. *IEEE Journal of Oceanic Engineering*, OE–11(4):449–461, 1986.

[2] D. R. Blidberg, R. M. Turner, and S. G. Chappell. Autonomous underwater vehicles: Current activities and research opportunities. *Robotics and Autonomous Systems*, 7:139–150, 1991.

[3] P. J. Feltovich, P. E. Johnson, J. A. Moller, and D. B. Swanson. LCS: The role and development of medical knowledge and diagnostic expertise. In W. J. Clancey and E. H. Shortliffe, editors, *Readings in Medical Artificial Intelligence*, pages 275–319. Addison–Wesley Publishing Company, Reading, Massachusetts, 1984.

[4] K. J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Inc. (Harcourt Brace Jovanovich, Publishers), New York, 1989.

[5] J. L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.

[6] J. L. Kolodner, R. L. Simpson, and K. Sycara-Cyranski. A process model of case-based reasoning in problem-solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985.

[7] D. McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[8] H. S. Shinn. Abstractional analogy: A model of analogical reasoning. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 370–387, Clearwater Beach, Florida, 1988.

[9] R. M. Turner. When reactive planning is not enough: Using contextual schemas to react appropriately to environmental change. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 940–947, Detroit, MI, 1989.

[10] R. M. Turner. *Adaptive Reasoning for Real-World Problems: A Schema-Based Approach*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.

[11] R. M. Turner and P. S. Eaton. Handling unanticipated events during collaboration. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 887–892, Atlanta, Georgia, 1994.

[12] R. M. Turner, P. S. Eaton, and M. J. Dempsey. Handling unanticipated events in single and multiple AUV systems. *l'Onde Electrique*, 74(5):36–42, September/October 1994. Published by Société des Electriciens et des Electroniciens (SEE).

[13] R. M. Turner and R. A. G. Stevenson. ORCA: An adaptive, context-sensitive reasoner for controlling AUVs. In *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology (UUST '91)*, pages 423–432, 1991.

[14] D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3), 1984.