CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Calhoun, Institutional Archive of the Naval Postgraduate School

**Calhoun: The NPS Institutional Archive**

Faculty and Researcher Publications | Faculty and Researcher Publications Collection

1996-03

# The Rational Behavior Software Architecture for Intelligent Ships : An Approach to Mission and Motion Control

Byrnes, R.B.

Byrnes, R.B., Healey, A.J., McGhee, R.B., Nelson, M.L., and Kwak, S.H., "A Rational

http://www.nps.edu/library

Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

**Ronald B. Byrnes, Anthony J. Healey,
Robert B. McGhee, Michael L. Nelson, Se-Hung Kwak,
and Donald P. Brutzman**

# The Rational Behavior Software Architecture for Intelligent Ships

## *An Approach to Mission and Motion Control*

**ABSTRACT** The solutions to the power projection, transportation, and operational needs of the Navy as it faces the 21st century must account for reduced manning levels. This leads naturally to increased use of computers, automation, and intelligent systems in the concept and design of the next generation of ships. In addition to the acknowledged hardware needs, the problem of autonomic and autonomous control of shipboard systems and missions are amenable to and will, in fact, require software solutions. Despite current technology, large, reliable software systems are difficult to achieve because correctness in requirements analysis, design, implementation, testing, modification, and maintenance of software are difficult. Software is also difficult to quantize and display; hence, the effort and costs involved in its development are easily underestimated. This paper describes an approach to the problem of providing structure, in the form of a software architecture, to the software performing autonomous control of missions and their related tasks. In concert with the need to reduce complexity, the architecture must support simple, rapid reconfiguration of code should vehicle capabilities or mission requirements change. Building upon recent efforts with control of Autonomous Underwater Vehicles (AUVs), we propose a tri-level control system architecture called the Rational Behavior Model (RBM) as an approach to autonomous and autonomic control of surface ship missions and systems.

## Introduction

Since 1987, the Naval Postgraduate School has been involved in research into advanced control concepts for robotics as applied to Autonomous Underwater Vehicles (AUVs). Early on, it was perceived that the future needs of the US Navy would require more use of autonomous and autonomic systems, especially where increased reliance on onboard computer-based decision making was required to meet requirements for decreased response times and overall reduced manning. While we have been concentrating on the uses of AUVs in a mine hunting mission scenario [33,36], the general structure and capabilities of an intelligent mission level controller for an AUV would have similar application and use for a future intelligent ship.

In either case, a particular concern is the requirement to join mission level decision making with modular task (or behavior) level coordination and finally low level vehicle subsystem control. These levels are well suited to an overall software structure, called a *software architecture*, in which the control problem is viewed from different levels of abstraction. Two notable earlier proposed software architectures for vehicle control are described in [34,35].

The Rational Behavior Model (RBM) is based on three levels of abstraction, called the Strategic, the Tactical, and the Execution levels, respectively. The first of these, conceptually the "top" level, contains a rule-based strategy for the accomplishment of the mission, contingent on variations in the internal, external, and operational environments. The intermediate tactical level consists of objects, in an object-oriented software sense, which represent the natural division of operations in the staff of a manned ship. Included herein are the navigator, engineer, weapons officer, and sonar section, all under the management of the Officer of the Deck (OOD). The lowest level of the model, the execution level, provides the data and control required by the automated servo systems of the vehicle that are responsible for the basic mobility of the vehicle and its controllable Hull, Mechanical, and Electrical (HM&E) subsystems.

This paper presents a brief description of the Rational Behavior Model and the constraints placed on its implementation followed by a discussion of an instantiation for a specific vehicle with a particular mission. Finally, lessons learned and plans for future research are presented. While our work has been concerned with control of autonomous underwater vehicles, we strongly believe that RBM has direct applicability to the problems faced by the surface ship community.

## The Rational Behavior Model

The Rational Behavior Model is a tri-level architecture for the intelligent control of autonomous and autonomic vehicles. For the purpose of this paper, we define

autonomic as "having an involuntary reflexive capability" and autonomous as "responding and reacting independently" [22]. RBM uses the principle of abstraction to simplify the problem of mission control, and as such represents a specialization of the SSS architecture (symbolic, subsumption, servo) [1]. In particular, in RBM, the top level is entirely symbolic and has no global variables, the bottom level is synchronous and entirely numerical, and the middle level provides an asynchronous interface between the other two levels. This division reflects the approach to the control problem typically employed on manned vehicles, such as surface ships. In these cases, control is viewed from different levels of abstraction in which the highest level (the commanding officer or Captain) designates and sequences goals through a deliberative process, the lowest level (the crew) operates vehicle actuators and sensors in response to commands, and the middle level (the commander's staff) decomposes tasks so as to produce commands to the Execution level in support of goal achievement.

We believe that implementation of the RBM architecture is best accomplished with programming languages and operating systems tailored to each level [2]. In particular, imperative languages with multitasking operating systems utilizing timed interrupts are preferred for the bottom level, rule-based languages are preferred for the top level, and languages supporting object hierarchies and event-driven multitasking are preferred for the middle level. These choices naturally fit the backgrounds and education of mission specialists at the top, computer scientists in the middle, and control engineers at the bottom. This facilitates the team development and modification of large software systems and promotes the reuse of software modules across vehicles and applications. Current commercial off the shelf (COTS) programming languages, operating systems, and computing hardware platforms are adequate for effective realization of RBM systems. The major characteristics of all three levels of RBM are summarized in Table 1.

TABLE 1

**Characteristics of RBM (from [4])**

**Strategic Level**
- Symbolic computation only; contains mission doctrine/specification
- No storage of internal vehicle or external world state variables
- Rule-based implementation, incorporating rule set, inference engine, and working memory (if required)
- Non-interruptible, not event-driven
- Directs the Tactical level through asynchronous message passing
- Messages may be either commands or queries requiring YES/NO responses
- Operates in discrete (Boolean) domain independently of time
- Building block: the goal
- Abstraction mechanisms: goal decomposition (RBM-B) and rule partitioning (RBM-F); both based on goal-driven reasoning

**Tactical Level**
- Provides asynchronous interface between Strategic and Execution levels
- Behaviors (tasks) reside here and may execute concurrently
- Behaviors are implemented as methods of objects
- Primitive goals activate one or more behaviors
- External interface of the model consists of two parts: the behavior activations from the Strategic level and the command/telemetry paths to/from the Execution level
- Workd and Mission models maintained here
- Responds to Strategic level with logical TRUE/FALSE
- Setpoints, modes, active sensor commands, and non-routine data requests are output to the Execution level
- Not interruptible except for data transfers; hard deadlines cannot be guaranteed
- Operates in discrete event/continuous time domains
- Building block: objects with behaviors
- Abstraction mechanisms: class and composition hierarchies

**Execution Level**
- Numeric processing only
- Responsible for software to hardware interface, underlying vehicle stability
- All synchronous (hard real-time) processes reside at this level
- Sensor data processed to specification of Tactical level
- Servo loops run continuously and concurrently, synchronized by timed interrupts
- Operates in continuous space/time domains
- Building block: servo loops and signal processing algorithms
- Abstraction mechanisms: loop composition, sampling frequency, and data smoothing

## STRATEGIC LEVEL

The commander of an autonomous/autonomic ship must have an effective means of expressing the desired mission to the vehicle, along with procedures for the replanning of missions due to the unrecoverable degradation of one or more subsystems. It is the Strategic level that addresses this need by encapsulating the explicit, high-level logic required to perform these activities. Indeed, it was in response to the concern of users that an intelligent vehicle behave rationally, that the name Rational Behavior Model was chosen. The basis for the Strategic level design is the top-down decomposition of the mission based on goal-directed reasoning. This process involves the successive refinement of a root goal into constituent subgoals, continuing until simple, primitive goals are identified. When goals are not amenable to further simplification, direct implementation via messages to the Tactical level occurs. We have used mostly goal decomposition in Prolog for the Strategic level. However, we have also had success with rule partitioning in the C-based forward chaining language Clips [3,4,5].

For the purpose of autonomous and autonomic vehicle control, the Strategic level incorporates mission-level control logic. With respect to this level, the following restrictions are imposed:

(1) The Strategic level is based on goal-driven (top down) rather than data-driven (bottom up) reasoning.

(2) The Strategic level contains no state other than the state of the reasoning process itself. World models, vehicle states, and numerical mission parameters are maintained by the Tactical level. The purpose for this is to support determinism and hence predictable (rational) responses from the vehicle.

(3) The Strategic level is implemented or specified in a commercially available rule-based language.

(4) As in SSS, the Strategic level operates asynchronously in discrete (boolean) space. However, we further specify that it is non-interruptible. That is, the Strategic level is explicitly not event driven. Rather, it obtains information from the Tactical level by polling during mission execution and selects paths of reasoning based on this information.

These restrictions are derived from our practical experience in the development of autonomous vehicle control software [33]. As such, these conditions allow us to use a commercially available Prolog interpreter (Quintus Prolog [32]) to realize a backward chaining [6] version of the Strategic level [3]. Because Prolog directly implements depth first search of a dynamic AND/OR goal tree [6,7] and has a very clean syntax for the description of such trees [8], this language is very appropriate for the development of top level control software.

While RBM uses goal-driven reasoning at the Strategic level, it does not follow that a backward chaining language is required for this purpose. An attractive alternative is to use Prolog as a specification language, which could be either translated into some other language or compiled into a corresponding finite state machine [5,9,10]. We have ourselves accomplished manual translation of Prolog into Lisp for walking machine control [10]. We have also translated Prolog into CLIPS, a forward chaining language, as an alternative means to implement the Strategic level for our AUV [3,5].

## TACTICAL LEVEL

The Tactical (middle) level acts as an interface between the knowledge-based Strategic level and the hardware-controlled subsystems of the Execution level. In object-oriented terms, we have implemented the behaviors of the Tactical level as methods of software objects. These methods may call other methods at the Tactical level or send commands to the Execution level. The Tactical level is also responsible for data collection during the mission and for contingency planning should the need arise. Hence, detection of and automatic recovery from machinery faults not affecting the overall mission are accomplished at this level. The Tactical level also maintains the numerical aspects of the mission model and internal and external world models.

The Tactical level of RBM manages the interface between the goals specified by the Strategic level and the actions performed by the Execution level. To this end, the following attributes characterize the Tactical level:

(1) As in the definition of SSS, the Tactical level operates in a discrete event space and in continuous time. That is, at this level, decisions are made in response to queries or commands which can arrive from the Strategic level at any time. On the other hand, the Tactical level receives information from the Execution level only on a timed interrupt basis. Thus, it is not directly triggered by events occurring at the Execution level, but nevertheless may at any time detect an event based on stored data from the Execution level.

(2) Outputs from the Tactical level to the Execution level are of three types: discrete mode changes, non-routine data requests, and continuous set points [11].

(3) To enhance modularity and maintainability, and to provide precise accepted terminology, the Tactical level is implemented as a software object hierarchy [9,12,13,14]. The methods [12] of these objects constitute the behaviors of this level. Child (dependent) objects are components of their parent objects and can be accessed only by parent methods. The single root object of the hierarchy is responsible both for asynchronous communications with the Strategic level and for transmission of orders to the Execution level, thereby avoiding the issuance of conflicting vehicle control commands.

(4) Each input to the Execution level comes from just one object at the Tactical level. This constraint ensures that competition for vehicle resources is resolved at the Tactical level.

(5) In order to facilitate portability and understandability of code, we prefer that concurrency at the Tactical level, when needed, be supported by the programming language rather than solely by the operating system. In light of condition (3) above, we also believe that the programming language selected for implementation of the Tactical level should be at least object-based, and preferably object-oriented [12,13].

Having imposed all of the above conditions on the Tactical level, it becomes difficult to find a suitable language. In [10] and [14], we used CLOS, the object-oriented facility of Common Lisp [15]. This means that, in this example, our Tactical level allowed for no concurrency. It is our belief that this succeeded because the vehicle under study (the ASV walking machine) actually implemented supervisory control [16], with a human operator in the loop, thereby limiting the use of RBM to leg coordination, a relatively simple function compared to autonomous mission control. In [3], we implemented the Tactical level in C, which has neither objects nor tasking constructs. This made coding rather difficult and, as of now, we have aban-

doned C and have instead chosen Ada [17,18] and C++ [5] as the languages of choice. Ada is, of course, the Department of Defense standard language for embedded systems, and possesses both objects and tasking capabilities. Furthermore, with the introduction of Ada 95 [37], it includes the notion of class inheritance, making it a true object-oriented language. Because Ada 95 was not yet available to us, we obtained this latter feature through an Ada extension called Classic-Ada [19].

### EXECUTION LEVEL

The Execution (bottom) level is responsible for controlling the machinery, sensors, and control surfaces. It directly controls the vehicle's heading and speed in response to commands from the Tactical level. It also sends sensory information back to the Tactical level and is the final chance for vehicle safety, so commands that would otherwise endanger the vehicle are overridden here. For example, the vehicle would execute reflexive evasive maneuvers rather than run into an obstacle. Similarly, other autonomic actions, such as a "flinch" prior to receiving an incoming missile, are implemented here. Most of the algorithms used at this level have their basis in modern control theory and as such involve strictly numerical computation.

This level, called the "servo" level in [1], and the "execution" level in [20], is certainly the best understood of the three levels of RBM. Indeed, despite the existence of many important problems at this level such as rudder roll stabilization [21], it is often taken more or less for granted by researchers concentrating on the upper levels of control.

While our research to date has been concerned principally with fully autonomous vehicles [24], it is important to realize that a continuous spectrum of human involvement in sparsely manned vehicle control is possible. That is, human control can range from no on-line interaction (complete autonomy), through supervisory control [16], to direct control of a subsystem. Thus, we believe that at least the following functions should be provided at the Execution level for any autonomous/autonomic vehicle employing the RBM architecture:

(1) steering autopilot for heading control (heading mode) or for yaw rate control (rate mode), including rudder roll stabilization where effective.

(2) a speed control autopilot, including integrated propulsion control, to adjust the vehicle speed on command, either in vehicle speed control mode or propeller rate/pitch control mode.

(3) integrated damage control systems.

(4) integrated machinery monitoring and control.

Of course, the achievement of this software functionality is dependent upon the existence of a local area network connecting all major subsystems throughout the ship.

In addition to insuring basic vehicle stability of the vehicle, the Execution level of the vehicle control system also includes the operation of sensing systems and sufficient data processing to provide interpreted data to the Tactical level for situation assessment. Thus, for example, it is important to determine how much sonar data must be made available within the framework of the vehicle autopilot update rate so that a reflexive capability for obstacle avoidance resides within this level [24].

Due to the numerical nature of the computations associated with these systems, the Execution level of RBM is written in an imperative programming language. We have used C for our experiments, but C++ and Ada provide viable alternatives. Of course, when using C or C++, it is necessary to relegate tasking to the operating system, since neither language provides this feature.

While asynchronous multitasking constitutes an important research area at the Tactical level, it appears to us that processes at the Execution level need not be event driven, but rather can run on a fixed schedule triggered by a timed interrupt from a real-time clock. Rate monotonic scheduling guarantees efficient use of processor capacity in such circumstances [28].

## Implementation of the Model

The major portion of RBM is mission independent. In fact, many software components developed for one system can be shared among a wide variety of vehicles with minimal modification [29]. However, in order to instantiate a complete, correctly working RBM architecture, a specific mission and vehicle must exist. In this paper, the test mission is the *Florida mission*, so-called because demonstrations were originally scheduled to take place off the Florida coast [30]. In addition, the Naval Postgraduate School (NPS) "PHOENIX" AUV is chosen as the target vehicle [33]. This section will discuss the specific implementation of RBM used for this configuration. While we are not proposing that a surface ship be fully autonomous with respect to communications and control, much of the discussion that follows is pertinent to the unmanned portions of the autonomic ship concept.

All on-board computer hardware, languages, and operating systems for the PHOENIX AUV were dictated by practical considerations. Specifically, Ada and Prolog are not available for the real-time operating system OS/9, while DOS does not support the multitasking features we desire in our Execution level software, presently written in C. We believe that situations like this will arise frequently in the development of RBM software for autonomous and autonomic vehicles, and that heterogenous computers therefore represent an effective type of host. On the other hand, there is no reason why a homogeneous distributed system could not also be used, if available. In fact, our development environment is homogeneous in that it uses three Unix workstations, one for each level of RBM, communicating over an ethernet local area network

[4]. The following discussion reflects a version of RBM applicable to either scenario.

## PHOENIX AUV STRATEGIC LEVEL

The Strategic level software of RBM is divided into two sections: a mission-specific part called the *Mission Specification* and a mission-independent part called the PHOENIX AUV *Doctrine*. For this study, the Mission Specification implements the Florida mission while the Doctrine contains rules specific to the operation of the PHOENIX AUV. Figure 1 contains the implemented Strategic level written in the Prolog programming language.

Each Prolog rule follows the general format of Figure 2 and represents an *if-then* relation. The rule is divided into a head and a body. The head of a rule corresponds to the *then* part of the rule and the body part is equivalent to the *if* part. With respect to mission accomplishment, the Prolog rule can be interpreted as a goal decomposed into constituent subgoals. Therefore, if all subgoals are satisfied, then the corresponding goal at the left of ".-" is satisfied. A characteristic of Prolog syntax is to relate expressions separated by a comma through the logical AND operator. If a logical OR relationship exists between subgoals, the OR is expressed by writing multiple Prolog rules with identical heads. In Figure 1, the two "initialize" rules represent such a relationship. In this way, the "initialize" goal can be achieved by either accomplishing the subgoals in the first rule or satisfying the subgoals in the second rule.

An important characteristic of the Strategic level of RBM is that an explicit sequence of goal achievement defines mission success or failure. Prolog always attempts to satisfy a subgoal by matching it to a fact (essentially a rule without a body) or a rule head in textual order; i.e., from top-to-bottom (for OR-related rules) and left-to-right (for AND-related subgoals). When a subgoal-rule head match is found, the search process proceeds to the first subgoal in the matched rule and another match is attempted. The algorithm guiding that search, called the *inference engine*, marks each goal to provide a reference should the current inference chain fail. If a match cannot be made given the existing circumstances, an attempt is made to resatisfy the most recent successful subgoal through a control mechanism called *backtracking* [31]. If no subgoal can be satisfied, the corresponding rule is skipped and an alternative rule is selected, if available. Rule and subgoal placement are therefore critical if the proper response from the AUV is to be achieved.

In the Prolog code of Figure 1, a subset of the full features of Prolog is utilized to suit the restriction that the Strategic level contain no storage of internal vehicle or external world state variables. Thus, Prolog clauses are used as rules without the application of the unification feature of Prolog. As a result, the Prolog rule heads do not contain variables. This greatly simplifies the modification of code resulting from mission reconfiguration and prevents the introduction of undesired side effects which characterize software systems employing global data structures.

Prolog provides the built-in control primitive "repeat" which, when used in concert with backtracking, allows for the creation of loops. When first encountered, the repeat predicate succeeds and the loop is entered. Repeat subsequently succeeds when encountered through backtracking. This provides for multiple attempts to satisfy those subgoals lying to the right of the repeat. Another control primitive required to insure the strict, iterative execution of the loop is the cut, denoted by "!", which acts to block backtracking. In the context of RBM, the cut is used to eliminate unnecessary search paths and to force a specific sequence of subgoal testing [31].

The program in Figure 1 is initiated when the query "?-execute_auv_mission." is issued to the Prolog inference engine. Scanning the heads of each rule starting from the top of the rule set, the rule "execute auv mission :- initialize, repeat, mission." is encountered. Prolog will first attempt to satisfy the subgoal "initialize". After marking this subgoal, the rule set is again scanned from the top in an attempt to find a matching rule head. A match is made with the first "initialize" rule. The first subgoal of this rule, "vehicle_ready_for_launch_p(ANS1)," is then encountered. This subgoal is a primitive goal in that it cannot be decomposed any further. When the Strategic level reaches such a primitive goal, it generates either a predicate query or a command to the Tactical level. A predicate query expects a TRUE/FALSE response from the Tactical level, and the returned value influences the subsequent reasoning path of the inference engine. A command, on the other hand, is a directive that initiates an action in the Tactical level. The primitive goal "vehicle ready_for_launch" is a predicate query, because its argument ANS1 is bound to TRUE or FALSE by the Tactical level. This value is then determined at the Strategic level through the test "ANS1 = = 1". If the value of ANS1 is 1 (representing TRUE), then "ANS1 = = 1" succeeds. The next subgoal, "select_first_waypoint(ANS2)" is then reached. This primitive subgoal is an example of a command, and as such, directs the Tactical level to select the first waypoint from the list of waypoints maintained at the Tactical level.

If, on the other hand, the value of ANS1 is 0 (representing FALSE), "ANS1 = = 1" fails. In this case, the Prolog inference engine initiates backtracking and tries to re-satisfy the subgoal "vehicle_ready_for_launch_p(ANS1)". However, this attempt fails because there is no other way to satisfy the "vehicle_ready_for_launch_p(ANS1)" subgoal. Consequently, the first "initialize" rule fails, and the second "initialize" rule is invoked, resulting in mission termination.

This process continues in similar fashion for the remaining rules in an attempt to satisfy the original query.

```
* --------MISSION SPECIFICATION FOR SEARCH AND RESCUE--------- */

initialize :- vehicle_ready_for_launch_p(ANS1),ANS1 == 1, select_first_waypoint(ANS2).
initialize :- alert_user(ANS), fail.

mission :- in_transit_p(ANS1), ANS1 == 1, transit, !, transit_done_p(ANS2), ANS2 == 1, fail.
mission :- in_search_p(ANS1), ANS1 == 1, search, !, search_done_p(ANS2), ANS2 == 1, fail.
mission :- in_task_p(ANS1), ANS1 == 1, task, !, task_done_p(ANS2),ANS2 == 1, fail.
mission :- in_return_p(ANS1), ANS1 == 1, return, !, return_done_p(ANS2), ANS2 == 1, wait_for_recovery(ANS3).

transit :- waypoint_control.
transit :- surface(ANS1), wait_for_recovery(ANS2).

search :- do_search_pattern(ANS), ANS == 1.
search :- surface(ANS1), wait_for_recovery(ANS2).

task :- homing(ANS1), ANS1 == 1, drop_package(ANS2), ANS2 == 1, get_gps_fix(ANS3), ANS3 == 1,
        get_next_waypoint(ANS4), ANS4 == 1.
task :- surface(ANS1), wait_for_recovery(ANS2).

return :- waypoint_control.
return :- surface(ANS1), wait_for_recovery(ANS2).

/* ---------- NPS AUV DOCTRINE ----------*/

execute_auv_mission :- initialize, repeat, mission.

waypoint_control :- not(critical_system_prob), get_waypoint_status, plan, send_setpoints_and_modes(ANS).

get_waypoint_status :- gps_check, reach_waypoint_p(ANS1), ANS1 == 1, get_next_waypoint(ANS2).
get_waypoint_status.

gps_check :- gps_needed_p(ANS1), ANS1 == 1, get_gps_fix(ANS1).
gps_check.

plan :- reduced_capacity_system_prob, global_replan.
plan :- near_uncharted_obstacle, local_replan.
plan.

near_uncharted_obstacle :- unknown_obstacle_p(ANS1), ANS1 == 1, log_new_obstacle(ANS2).

local_replan :- loiter(ANS1), start_local_replanner(ANS2).

global_replan :- loiter(ANS1), start_global_replanner(ANS2).

critical_system_prob :- power_gone_p(ANS), ANS == 1.
critical_system_prob :- computer_system_inop_p(ANS), ANS == 1.
critical_system_prob :- propulsion_system_p(ANS), ANS == 1.
critical_system_prob :- steering_system_inop_p(ANS), ANS == 1.

reduced_capacity_system_prob :- diving_system_p(ANS), ANS == 1.
reduced_capacity_system_prob :- bouyancy_system_p(ANS), ANS == 1.
reduced_capacity_system_prob :- thruster_system_p(ANS), ANS == 1.
reduced_capacity_system_prob :- leak_test_p(ANS), ANS == 1.
```

**FIGURE 1. The Florida Mission in Prolog**

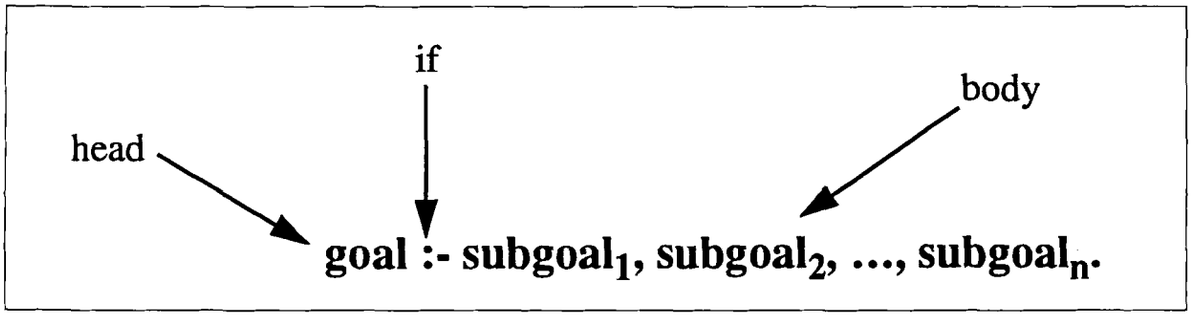


FIGURE 2. A Prolog Rule

This particular rule set is collectively referred to as the Florida mission and consists of four distinct phases called "transit", "search", "task", and "return". Each phase has two rules associated with it. The first rule specifies the sequence of goals which occur in normal circumstances. A second rule is included for each phase to act as a software fail-safe or "exception" should a condition arise resulting in the failure of the current phase. For this mission, each fail-safe consists of the two commands "surface(ANS1)" and "wait_for_recovery(ANS2)". Post-mission analysis of telemetry and status data, continuously recorded by the Tactical level, would presumably yield information about the cause of the failure. Again, the nominal rule is placed textually before the fail-safe rule so as to insure that the nominal rule is attempted first.

## PHOENIX AUV TACTICAL LEVEL

The Tactical level is composed of software objects that communicate via message passing. A significant provision is that some (or all) of the objects may be active at a given time; that is, several objects may embody separate, distinct threads of control. On a single processor, logical concurrency of these objects is realized through the "interleaving" of each task's execution under the guidance of a time-sharing or priority-based algorithm. If multiple processors are available to support true parallel execution, objects may run simultaneously. In either case, the actions of each are coordinated through the sending of messages to one another. Concurrency may be provided in several ways, including control constructs provided by a concurrent programming language. This allows for the explicit identification of potential parallelism within the program. This was one of the reasons that we chose Ada for this implementation. Although we have not yet used this feature, it is expected that future extensions will call for concurrent processing.

The Tactical level developed for the PHOENIX AUV is shown in Figure 3. Each block in the diagram represents a distinct entity and corresponds to a software object. Most of the objects rearranged into a (composition) hierarchy, as indicated by the solid lines linking them together. The AUV Officer of the Deck (OOD) resides at the top of the hierarchy and assumes overall control of the operation. In addition, the OOD provides the single interface between the Strategic and Tactical levels. Primitive goals from the top level are passed to the OOD who in turn activates behaviors within the Tactical level designed to satisfy those goals. Returning to the analogy of the manned crew, the Captain of the submarine (the Strategic level) issues commands to, or asks for, status reports from the OOD (the root object in the Tactical level). The OOD then issues the appropriate orders to satisfy the goal or query presented by the Captain.

All the behaviors that are capable of being performed by the vehicle are embodied within the various objects of the Tactical level. The OOD must coordinate the actions of each object to ensure that each task is accomplished as expected. The behaviors, for their part, are reflected in the methods contained within the applicable object(s). When a behavior involves the interaction of multiple objects, communications are provided through the passing of messages. As depicted in the figure, direct communications between members of the hierarchy is restricted to parent-child links. While this comes at the expense of efficiency, the benefits include the avoidance of unconstrained communication paths and a greater degree of modularity. These characteristics support RBM's emphasis on providing a framework to the user that aids in the understanding and maintenance of the software at this level.

Communication with the Execution level is also restricted. Commands, in the form of packets containing numerical set points, non-routine data requests, and discrete mode changes, are issued only from the command sender object. Similarly, telemetry data from the Execution level is received solely by the sensory receiver object. By constraining these interfaces, command conflicts and data inconsistency are avoided.

Several objects in the Tactical level are not explicitly connected to the object hierarchy. These represent data stores (databases) intended to be accessed by any requesting object. The state of the mission, the environ-

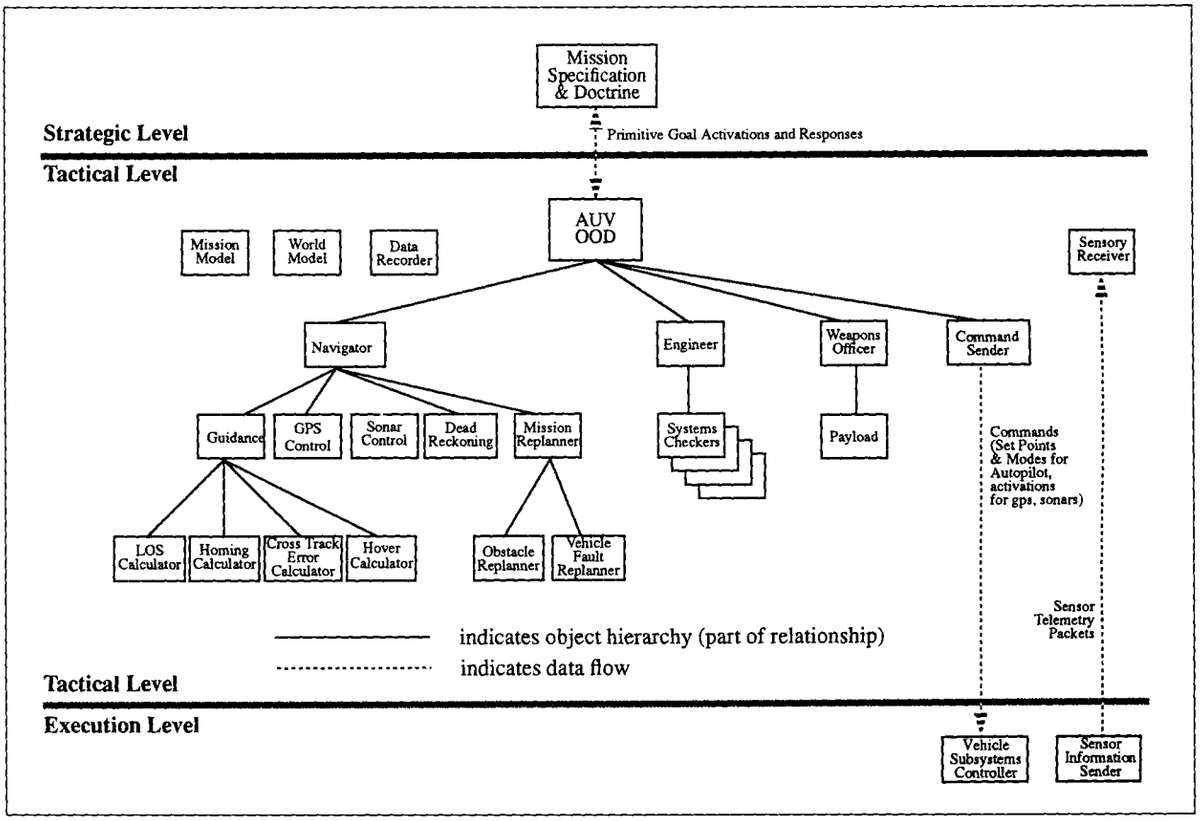


**FIGURE 3. Tactical Level Object Hierarchy for the NPS AUV Florida Mission (from [4])**

mental model, current sensor readings, and mission history are maintained and encapsulated within the corresponding object. Requests for information and data updates are handled as they arrive. Note that these objects do not directly participate in task accomplishment. Details of all objects in the Tactical level are given in [4,27].

### PHOENIX AUV EXECUTION LEVEL

RBM makes no contribution to this level. Most of the control concepts implemented here, such as sliding mode control, are well documented [23]. The current Execution level controls either the PHOENIX itself or a highly accurate real-time graphic simulator [36]. The language of choice for this level, based on its run-time efficiency, is C. Specifics of this and the PHOENIX AUV Integrated Simulator are discussed in the next section. Naturally, the application of RBM to surface ships would require that the complete implementation of appropriate execution level software be in place.

## Experimentation

The instantiation of RBM just described has been implemented in the laboratory on the PHOENIX AUV Integrated Simulator, a network consisting of an actual AUV computer system, a three-dimensional graphical simulation workstation, and appropriate support equipment [26]. The simulation experiments center around the search and rescue (Florida) mission, with the Strategic and Tactical levels of RBM hosted by the AUV computer and the Execution level residing on a Silicon Graphics Iris workstation. The findings of these experiments are summarized in the following paragraphs.

### SOFTWARE DEVELOPMENT ENVIRONMENT

While the development of RBM is simplified by its use of abstraction and separation of problem-solving responsibilities into three distinct levels, testing of the complete model is complicated by the expense of field testing and frequent nonavailability of the target vehicle due to hardware modification or rebuild. Integrated simulation pro-

vides a means of testing and evaluating vehicle control software despite these constraints. Readers interested in the concepts involved in this approach to AUV software development and testing are referred to [26].

Figure 4 portrays the configuration of the components of the Integrated Simulator used in the experiments to be discussed in this paper [25]. At the heart of the network is a Gespac card cage containing two microprocessor boards. The Gespac 80386, running DOS, hosts the Strategic and Tactical level of the PHOENIX AUV Rational Behavior Model. The Gespac 68030, running the OS-9 operating system, hosts the Execution level. The two processors are linked by a parallel and a serial connection, each designed to route data in a single direction; i.e., commands are passed from the Tactical to the Execution level over the parallel link, and telemetry data is sent in the reverse direction over a serial path. This telemetry data is received by the Execution level from the graphical workstation of the integrated simulator representing the vehicle and its operating environment (world model). The simulated vehicle's Execution level in turn receives messages from the Tactical level containing numerical set points, operational mode changes, and non-routine data requests. The 68030 and graphical workstation are linked to each other by an Ethernet connection.

Direct connections can be made between the Gespac 68030 and physical hardware components, as denoted by the solid arrow. In this way, sensors, actuators, and other vehicular subsystems can be tested in the lab prior to their installation. Finally, an additional serial port is available on the 68030 board which, when connected to an external monitor, allows for the creation, debugging, and modification of Execution level software. Furthermore, this port may be used to connect, via modem, a terminal collocated with the actual AUV. This allows for easy transfer and downloading of Execution level software and experimental data between the lab and test site.

The Gespac 80386 also provides a great deal of flexibility to the RBM software developer. An external EGA monitor is available for developing and testing the Strategic and Tactical level software. A dedicated modem is also available to allow for software development from remote sites removed from the lab. For experiments involving receipt and analysis of GPS data, a serial port with GPS receiver is available. The potential for parallelism at the Tactical level may be explored with an available transputer board.

In sum, the Integrated Simulator greatly facilitates the design, development, and integration of the many hardware and software components of the PHOENIX AUV. Each interface can potentially represent a source of bugs; however, these problems can often be detected and fixed in the lab, thus avoiding the expense and frustration of field test failure.

## THE SEARCH AND RESCUE MISSION

The search and rescue (Florida) mission provides an ideal test case for observing the global behavior of an autonomous vehicle and the capabilities of its control software architecture. Following an initialization sequence, the mission is composed of four phases: transit from the launch site to the search area by achieving a series of predetermined waypoints; performance of the search algorithm; execution of an appropriate task subsequent to locating the target; and returning or transiting to a final location. In the experiment, this scenario was implemented using a figure-8 path. The first half of the path constituted the transit phase. When the mid-point of the figure-8 was reached, the search and task phases were entered in succession. Neither the search nor the task phase was implemented in detail since the purpose of this exercise was to test the logical and behavioral aspects of the control architecture, and not specific algorithms or methods. The final phase of the mission, the return phase, required the attainment of waypoints making up the second half of the figure-8 path. Upon reaching the final goal, the AUV was then directed to secure its subsystems, surface, and await recovery.

The first test was designed to provide a basis for comparison and as such included no anomalies. The second test was identical to the first except that a failure in the vehicle's power subsystem was introduced midway through the transit phase. The third and fourth tests replicated the first two tests but utilized a forward chaining Strategic level implemented in CLIPS instead of the Prolog-based backward chaining version [3,4,5].

## RESULTS

Three classes of data were obtained for each test. First, visual observations of the graphical simulation were used to investigate the global behavior of the AUV manifested by the executing RBM. Second, time stamps were taken whenever a waypoint was attained. Third, a trace was taken which recorded the sequence of primitive goals achieved directed by the Strategic level of RBM. Figure 5 shows the PHOENIX AUV simulation in progress. White spheres represent waypoints and the dark sphere represents the final goal point (recovery site).

Visual observation of the simulation proved valuable in two important ways. The logic used by the Strategic level could be validated by comparing the resulting behavior of the vehicle with the requirements specified by the mission specialist. If the vehicle failed unexpectedly or performed an undesired action, faulty logic in the Strategic level or an erroneous method in a Tactical level object was indicated. Additionally, even though the vehicle performed as expected, an operational parameter beyond the control of RBM could have contributed to a failure state. For ex-

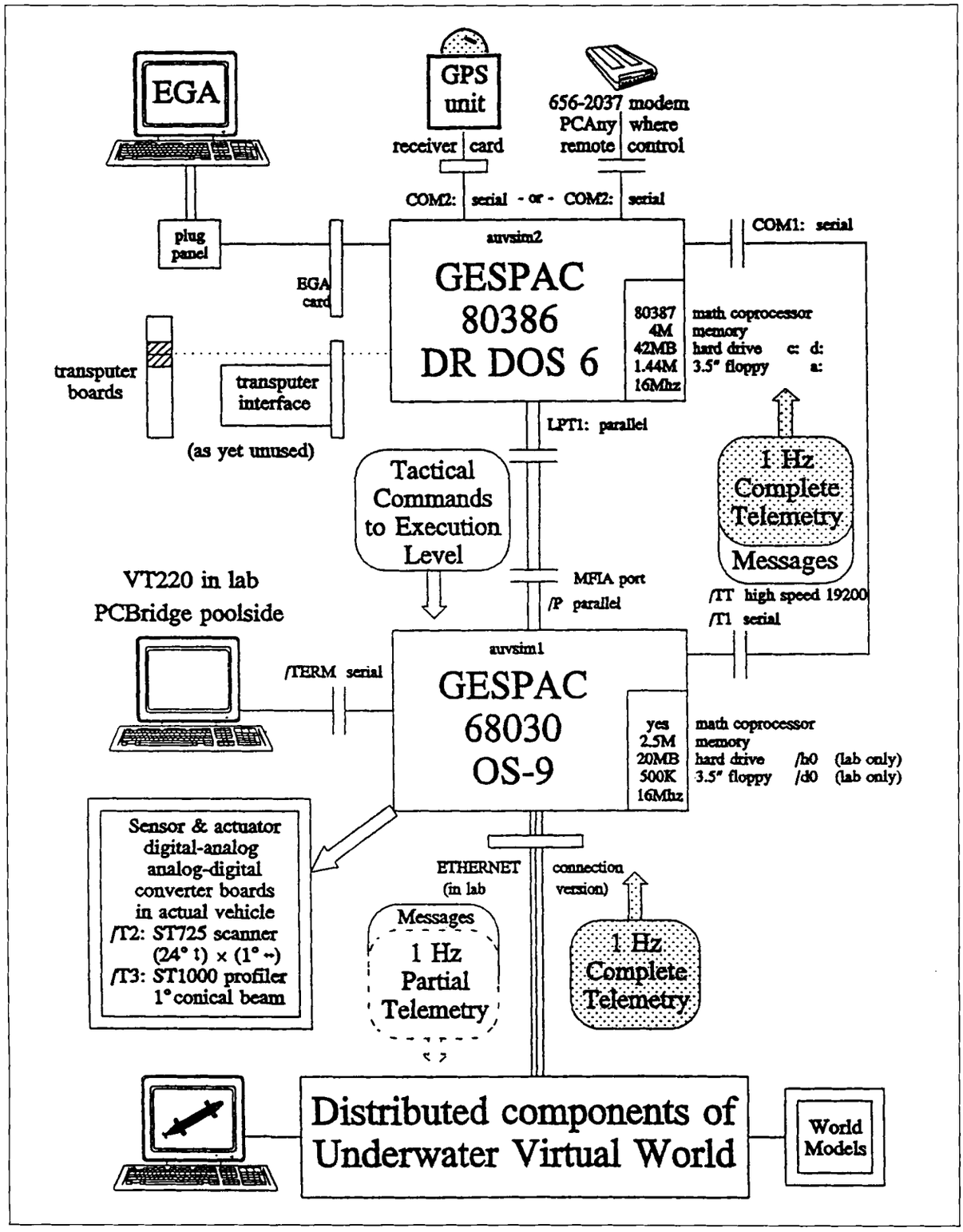


FIGURE 4. NPS AUV Integrated Simulator Configuration (from [25])

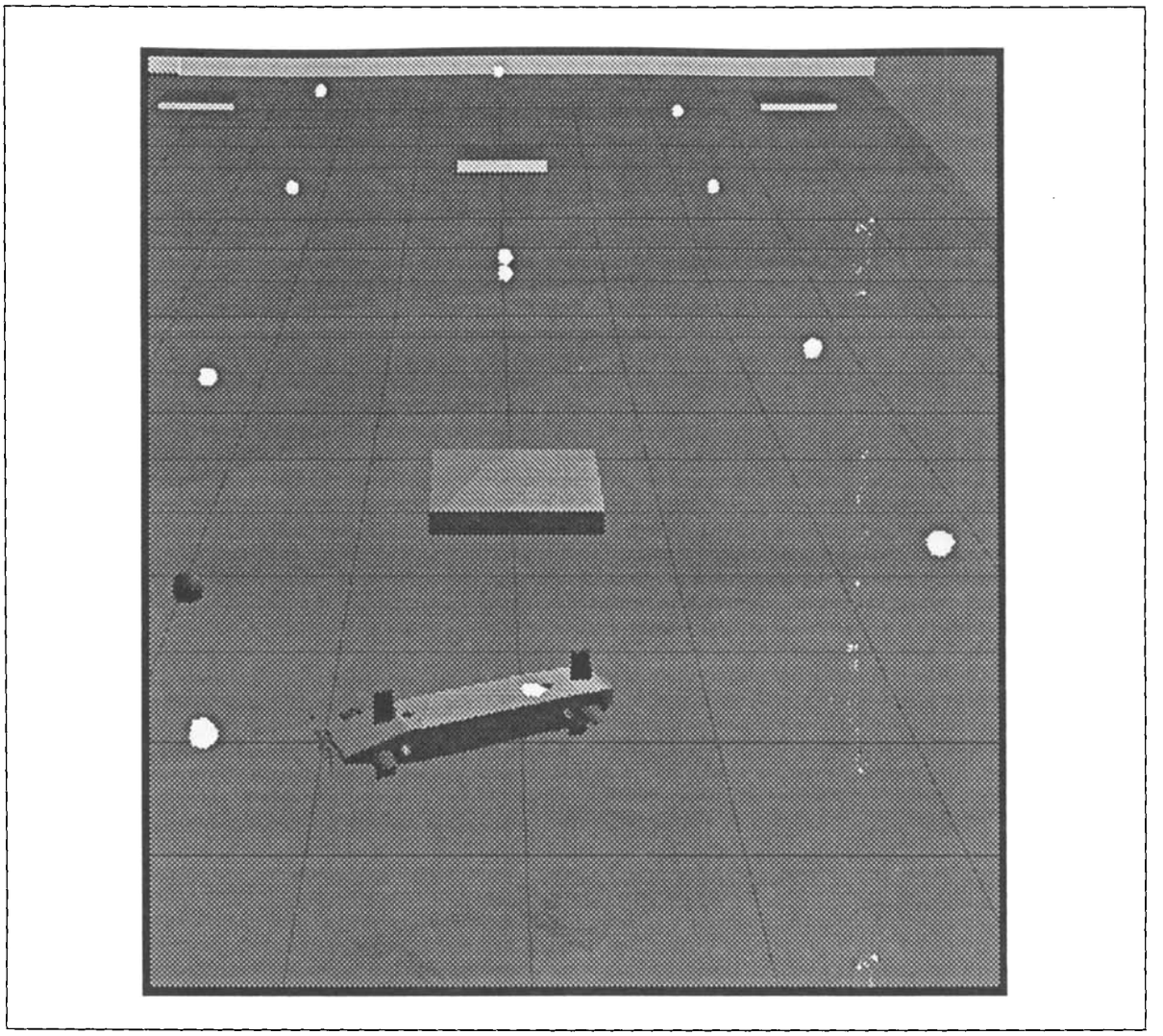

FIGURE 5. The NPS AUV Simulation

ample, if the vehicle speed is set at a value beyond some threshold, the AUV can be made to collide with a known object. These examples highlight the advantages of simulation testing prior to system integration.

A more precise check of Strategic level logic can be made with the execution traces. Each primitive goal, when successfully completed, is written to a file. Because the goals are listed sequentially, a chain of reasoning used by the Strategic level can be reconstructed.

The time stamped data was used to compare the relative execution speeds of the backward and forward chaining implementations of the Strategic level. These data, which are included in [4], were statistically identical, leading to the conclusion that both versions of the Strategic level performed sufficiently to satisfy the 2 Hz update rate required by the Execution level. This implies that the complexity of the Strategic level may increase up to the point at which the time taken by the Strategic level to reason about the next goal exceeds the minimum acceptable update rate between the Tactical and Execution levels.

## Conclusions and Future Research

From the results of this research, we have drawn several conclusions. Prolog, with its backtracking facility and textual ordering, is ideal for the specification of missions amenable to goal driven decomposition. Ada has proved to be

useful for the implementation of the Tactical level, because tasking and data encapsulation is provided by the language. C++ is being investigated for its object-oriented constructs, in combination with the real-time operating system VxWorks [38].

Future testing of RBM is scheduled to be done on the PHOENIX AUV and will involve extensions of the mission replanning logic at the Strategic level and automatic fault recovery at the Tactical level. This research will have direct applicability to all intelligent vehicles, autonomous, autonomic, or otherwise.

## ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation under Grant No. BCS-9306252. ✤

## REFERENCES

[1] Connell, J. H., "SSS: A Hybrid Architecture Applied to Robot Navigation," Proceedings of IEEE International Conference on Robotics and Automation, Nice, France, May 1992, pp. 2719-2724.

[2] Kwak, S. H., "A Computer Simulation Study of a Free Gait Motion Coordination Algorithm for Rough-Terrain Locomotion by a Hexapod Walking Machine," Ph.D. Dissertation, The Ohio State University, Columbus, Ohio, 1986.

[3] Byrnes, R. et al., "An Experimental Comparison of Hierarchical and Subsumption Software Architectures for Control of an Autonomous Underwater Vehicle," Proceedings of 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, Washington, D.C., June 2-3, 1992, pp. 135-141.

[4] Byrnes, R. B., "the Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA 93943, March 1993.

[5] Stroustrup, Bjarne, *The C++ Programming Language*, 2d ed., Addison-Wesley, 1991.

[6] Jackson, *Introduction to Expert Systems*, 2d Edition, Addison-wesley, 1990.

[7] Sanderson, A. C., Homem de Mello, L. S., and Zhang, H., "Assembly Sequence Planning," *AI Magazine*, Spring, 1990, pp. 62-80.

[8] Rowe, N. C., Artificial Intelligence Through Prolog, Prentice Hall, 1988.

[9] Simon, D., et al., "Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues," *IEEE Transactions on Control Systems Technology*, Vol. 1, No. 4, Dec. 1993, pp. 213-229.

[10] Kwak, S. H. and McGhee, R. B., "Rule-Based Motion Coordination for the Adaptive Suspension Vehicle in Ternary-Type Terrain," Technical Report NPSCS-91-006, Naval Postgraduate School, Monterey, CA, 93943, December 1990.

[11] Healey, A. J., et al., "Tactical/Execution Level Coordination For Hover Control of the NPS AUV II Using Onboard Sonar Servoing," Proceedings of the IEEE Symposium on Autonomous Underwater Vehicle Technology (AUV '94), Cambridge, MA, July 19-20, 1994.

[12] Booch, G., *Object Oriented Design*, Benjamin/Cummings, 1991.

[13] Miller, D. L. and Lennox, R. "An Object-Oriented Environment for Robot System Architectures," *IEEE Control Systems Magazine*, Vol. 11, No. 2, February, 1991, pp. 14-73.

[14] Kwak, S. H. and McGhee, R. B., "Rule-Based Motion Coordination for a Hexapod Walking Machine," *Advanced Robotics*, Vol. 4, No.3, November, 1990, pp. 263-282.

[15] Keene, *Object-Oriented Programming in Common Lisp*, Addison-Wesley, 1989.

[16] Yoerger, D. Neumann, J. and Slotine, J., "Supervisory Control System for the JASON ROV," *IEEE Journal of Oceanic Engineering*, Vol. 11, No. 3, 1986, pp. 392-400.

[17] Berzins, V. and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, 1991.

[18] Gehani, N., *Ada-Concurrent Programming*, Prentice Hall, 1984.

[19] *Classic-Ada User's Manual*, Software Productivity Solutions, Inc., Indiatlantic, FL, 1989.

[20] Wang, et al., "A Petri-Net Coordination Model of Intelligent Mobile Robots," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 4, July/August 1992, pp. 777-789.

[21] Powell, D. "Rudder Roll Stabilization—A Critical Review," Proceedings of the Ninth Ship Control Systems Symposium, Bethesda, MD, Sept. 10-14, 1990.

[22] *Webster's New Collegiate Dictionary.*

[23] Healey, A. J. and Lienard, D., "Multivariable Sliding Mode Control for Autonomous Diving and Steering of Unmanned Underwater Vehicles," *IEEE Journal of Oceanic Engineering*, Vol. 18, No. 3, July 1993, pp. 327-339.

[24] Healey, A. J., et al., "Research on Autonomous Underwater Vehicles at the Naval Postgraduate School," *Naval Research Reviews*, Vol. XLIV, No. 1, 1992, pp. 43-51.

[25] Brutzman, D. P., "Underwater Virtual World for an Autonomous Underwater Vehicle," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA 93943, Sept. 1994.

[26] Brutzman, D. P., "Integrated Simulation For Rapid Development of Autonomous Underwater Vehicles," Proceedings of 1992 IEEE Symposium on Autonomous Underwater Vehicle Technology, Washington, D.C., June 2-3, 1992, pp. 3-10.

[27] Nelson, M. L., et al., "Putting Object-Oriented Technology to work in Autonomous Vehicles," Proceedings of llth International Conference on Technology of Object-Oriented Languages and Systems (TOOLS '93), August 2-6, 1993, Santa Barbara, CA.

[28] Sha, L. and Goodenough, J. B., "Real-Time Scheduling Theory and Ada," *Computer*, Vol. 23, No. 4, April 1990, pp. 53-62.

[29] Byrnes, R. B., et al., "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Underwater Vehicles," Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH, Sept. 27-29, 1993, pp. 160-178.

[30] Steer, B., Dunn, S., and Smith, S., "Advancing and Assessing Autonomy in Underwater Vehicle Technology Through Inter-Institutional Competitions and/or Cooperative Demonstrations," Department of Ocean Engineering, Florida Atlantic University, Boca Raton, Florida, May 1992.

[31] Clocksin, W. F. and C. S. Mellish, *Programming in Prolog*, 2d Edition, Springer-Verlag, 1984.

[32] *Quintus Prolog Manual*, Release 3.1, Quintus Corporation, 1991.

[33] Healey, A. J., et al., "Evaluation of the NPS PHOENIX Autonomous Underwater Vehicle Hybrid Control System," Proceedings of the 1995 American Control Conference, Seattle, WA, June 21-23, 1995, pp. 2954-2963; also in Proceedings of the Use of Autonomous Vehicles in Mine Countermeasures Symposium, Monterey, CA, April 4-7, 1995, pp. 9-32 through 9-41.

[34] Albus, J. S., McCain, H. G., and Lumia, R., *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*, NIST Technical Note 1235, 1989.

[35] Miller, D. P., Slack, M. G., and Elsaesser, C., "An Implemented Intelligent Agent Architecture for Autonomous Submersibles," Proceedings of the American Society of Naval Engineers Intelligent Ship Symposium, Philadelphia, PA, June 1-2, 1994, TAB 12.

[36] McGhee, R. B., et al., "An Experimental Study of an Implemented GPS/INS System for Shallow-Water AUV Navigation (SANS)," Proceedings of the 9th International Symposium on Unmanned Untethered Submersible Technology, Durham, NH, September 25-27, 1995.

[37] *Ada 95 Reference Manual*, ISO/IEC/ANSI 8652:1995.

[38] VxWorks Promotional Guide, Wind River Systems, Inc., 1010 Atlantic Avenue, Alameda, CA.

**Ronald B. Byrnes** *is a U.S. Army Lieutenant Colonel commissioned into the Signal Corps in 1979. He has served in a variety of positions associated with tactical and operational communications at company, brigade, and corps level. L.C. Byrnes became a member of the Army Acquisition Corps in 1993 and is currently a computer scientist with the Software Technology Branch of the Army Research Lab. His current work includes continuation of his doctoral research into high-level control of autonomous vehicles; contributions to the Army's electronic records management and data standardization efforts; and automated support for flight plan verification. L.C. Byrnes received a B.S. degree in Mathematics from Midwestern State University in 1979, an M.S.E.E. degree from the Naval Postgraduate School in 1989, and a Ph.D. in Computer Science from the Naval Postgraduate School in 1993.*

**Anthony J. Healey** *graduated from London and Sheffield Universities with B.Sc. (Eng) and Ph.D. degrees in Mechanical Engineering in 1961 and 1966, respectively. He emigrated to the United States in 1966 and has taught at the Pennsylvania State University, Massachusetts Institute of Technology, and the University of Texas at Austin, becoming Full Professor of Mechanical Engineering in 1974. In 1981, he joined Brown and Root, Inc., as manager of the Pipeline and Subsea Technology Research Group. In 1986, he accepted the position as Professor and Chairman of Mechanical Engineering at the Naval Postgraduate School. His areas of specialty include mechanical system dynamics, vibration, and control systems, and he is currently the leader of an interdisciplinary project in mission planning, navigation, and control for Autonomous Underwater Vehicles at NPS.*

**Robert B. McGhee** *received a B.S. degree in Engineering Physics from the University of Michigan in 1952, and M.S. and Ph.D. degrees in Electrical Engineering from the University of Southern California in 1957 and 1963, respectively. From 1952 until 1955, he served on active duty as a guided missile maintenance officer with the U.S. Army Ordnance Corps. From 1955 until 1963, he was a member of the technical staff with Hughes Aircraft Company, Culver City, Ca., where he worked on guided missile simulation and control problems. In 1963, he joined the Electrical Engineering Department at the University of Southern California as an Assistant Professor, and was promoted to Associate Professor in 1967. In 1968, he was appointed Professor of Electrical Engineering and Director of the Digital Systems Laboratory at the Ohio State University. In 1986, he joined the Computer Science Department at the Naval Postgraduate School, where he served as chairman from 1988 until 1992. Since 1992, he has held a joint appointment as professor in the Computer Science and Electrical Engineering Departments at the Naval Postgraduate School. Dr. McGhee is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE).*

**Dr. Michael L. Nelson** *is currently on the faculty of the Computer Science Department of the Pan American University, Edinburg, Texas. He retired from the Air Force in 1995, where his last position was as chief of the Project/Configuration Management Branch, Information Systems Support Activity, headquarters, U.S. Commander in Chief, Pacific. He received his M.S. degree in Computer Science in 1984 from the Florida Institute of Technology and his Ph.D. in Computer Science in 1988 from the University of Central Florida. Dr. Nelson was previously on the faculty of the Computer Science Department at the Naval Postgraduate School. His primary research interest is object-oriented design techniques applied to real-time and robotic systems.*

**Se-Hung Kwak** *is currently the principal software engineer with Loral's Advance Distributed Simulation program with expertise in intelligent and autonomous agent control. He received his Ph.D. in Electrical Engineering at the Ohio State University in 1986, following a tour of duty as an Army Air Defense artillery officer in the Republic of Korea. Dr. Kwak was previously on the faculty of the Computer Science Department at the Naval Postgraduate School. He is a member of IEEE.*

**Donald P. Brutzman** *is a retired U.S. Navy officer currently on the faculty of the Operations Research Department of the Naval Postgraduate School. While on active duty, he served three submarine tours. Brutzman received a B.S.E.E. degree from the U.S. Naval Academy in 1978 and M.S. and Ph.D. degrees in Computer Science from the Naval Postgraduate School in 1992 and 1994, respectively. His research interests include 3D real time computer graphics, virtual worlds, scientific visualization, underwater robotics, distributed simulation, machine learning, and high-performance network applications. He is a member of IEEE, ACM and the American Association for Artificial Intelligence.*