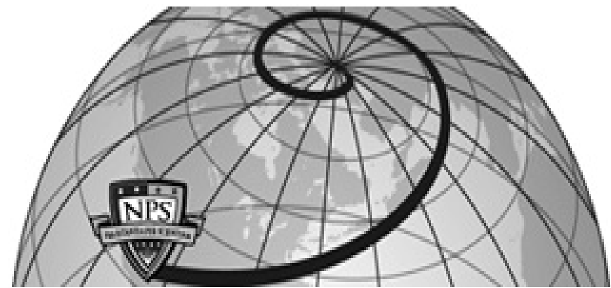
Calhoun
Institutional Archive of the Naval Postgraduate School



Calhoun: The NPS Institutional Archive

DSpace Repository

NPS Scholarship | Publications

1992

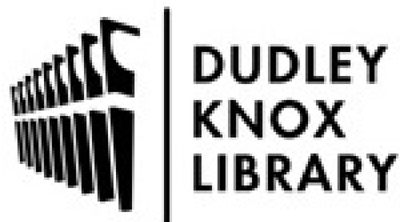
A Mission Executor for an Autonomous Underwater Vehicle

Lee, Yuh-jeng; Wilkinson, Paul

Lee, Yuh-Jeng, and Paul Wilkinson. "A mission executor for an autonomous underwater vehicle." (1992. NASA Technical Reports Server (NTRS).
https://hdl.handle.net/10945/50367

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun



DUDLEY KNOX LIBRARY

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

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

N92-16573

A MISSION EXECUTOR FOR AN AUTONOMOUS UNDERWATER VEHICLE

Yuh-jeng Lee and Paul Wilkinson

Computer Science Department
Naval Postgraduate School
Monterey CA 93943

Abstract. The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle (AUV). One facet of this research is to incrementally design a software architecture and implement it in an advanced testbed, the AUV II. As part of the high level architecture, a Mission Executor is being constructed using CLIPS version 5.0. The Mission Executor is an expert system designed to oversee progress from the AUV launch point to a goal area and back to the origin. It is expected that the Executor will make informed decisions about the mission, taking into account the navigational path, the vehicle subsystems health and the sea environment, as well as the specific mission profile which is downloaded from an offboard mission planner. Heuristics for maneuvering, avoidance of uncharted obstacles, waypoint navigation, and reaction to emergencies (essentially the expert knowledge of a submarine captain) are required. Many of the vehicle subsystems are modeled as objects using the CLIPS Object Oriented Language (COOL) embedded in CLIPS 5.0. Additionally, truth maintenance is applied to the knowledge base to keep configurations updated.

AUTONOMOUS UNDERWATER VEHICLE RESEARCH

The development of autonomous vehicles has been an ambition for decades. Automated weapons such as the Tomahawk missile now have a proven record of achievement in hazardous conditions. The MAZLAT/AAI Pioneer, a remotely-piloted vehicle (while not fully autonomous), similarly has a capable record in high-risk environments, as evidenced by the Gulf War. Several marine autonomous and remotely-piloted vehicles are already in use for such diverse functions as underwater cable inspection, hydrography, and mine-hunting. The practical advantage of low-risk to humans coupled with the potential ability to operate at over-the-horizon distances from the control platform make the autonomous underwater vehicle a highly desirable project. While there are several operational autonomous underwater vehicle testbeds in the United States, until recently most underwater vehicles have been tele-operated or merely data autonomous while receiving power via an umbilical cable.

Many software architectures have been proposed and are currently being tested for a fully autonomous underwater vehicle. One of the well-known is MIT's Sea Sprite Vehicle which adapted the layered control architecture proposed by Brooks (Bellingham 1990, Brooks 1986). The KB/EAVE (Knowledge-Based Experimental Autonomous Vehicle) AUV program of the University of New Hampshire's Marine Systems Engineering Lab essentially uses a subsumption architecture (as generally described by Brooks). High level and low level tasks are divided in hardware. The software uses the "focus of attention" approach to keep upper-level reasoning foremost while low-level behaviors occur (Blidberg 1990). International Submarine Engineering

of Canada also uses a layered control architecture with behaviors classified as reflexive, logical, and trained. These require reasoning on several levels, with planned and learned responses, encoded in a scripting language instead of a traditional AI language (Zheng 1990).

The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle. Both high-level and low-level software have gone through several versions of development. Currently, the software is destined to reside on a GESPAC MPU30HF processor board using the OS-9 operating system on a Motorola 68030 central processing unit. From a software architecture standpoint, the AUV software can best be designed in a hierarchical structure and viewed at different levels of abstraction for different purposes, for example, mission planning, mission execution, world modeling, collision avoidance, and vehicle control. This software has to perform both numeric computing and symbolic reasoning. Most of the computations also involve real-time constraints and time-dependent representations of the states of the AUV and the environment. In addition, many tasks are knowledge intensive and require domain specific information. For example, the collision avoidance routine needs to interpret sensor input, react to uncharted obstacles, replan a new vehicle path or mission based on available choices, and so on.

The NPS AUV II software is partitioned into several main modules, including an off-line mission planner, mission executor, guidance system , autopilot system, navigation system, sonar data processor, on-board mission replanner, and vehicle system monitor (Healy et al. 1990). Each of the modules contains submodules performing more specific tasks. For example the autopilot system includes routines for digital-analog data conversion, for hydrodynamic surfaces control, and for main motor control; the guidance system includes a local path planner for creating postures form waypoints and the tracking controller for providing desired postures to the autopilot (Cloutier 1990, Lee et al. 1991). Figure 1 shows the dataflow diagram of the AUV II baseline system.

DESIGN OF SKIPPER

The high-level design of the AUV II is the result of an incremental development which began in 1988 with AUV I. Initially, vehicle control was essentially lower-level closed-loop. Evolutionary changes in subsequent software designs resulted in the need for a high-level control module to coordinate the functionalities of various subsystems. The Mission Executor, SKIPPER, attempts to do this while integrating decisions based on input from three worlds: the vehicle's internal systems, the environment, and the mission. The design of Mission Executor essentially consists of a rule base and an object base. The major equipments aboard the submarine are modeled as objects to be monitored. Further, each obstacle encountered by the submarine sonar, whether planned for or not, is modeled as an object. Decisions on courses of action to take are modeled as objects for the purpose of easy retrieval via hyperlinks (this will be more self-evident shortly). All of these are linked together in the SKIPPER's Display, a blackboard subset of all of these. The SKIPPER's Display is a composite of the most vital information and consists only in the current decisions, obstacles which are still active (those in a 180-degree arc about the bow of the submarine), and the current state of the system monitors. This intelligent database is frequently updated and queried by the rule base.

While the vehicle's different states are updated and monitored by querying objects (and firing attached daemons), the heuristics for the three worlds (internal, environmental and mission)

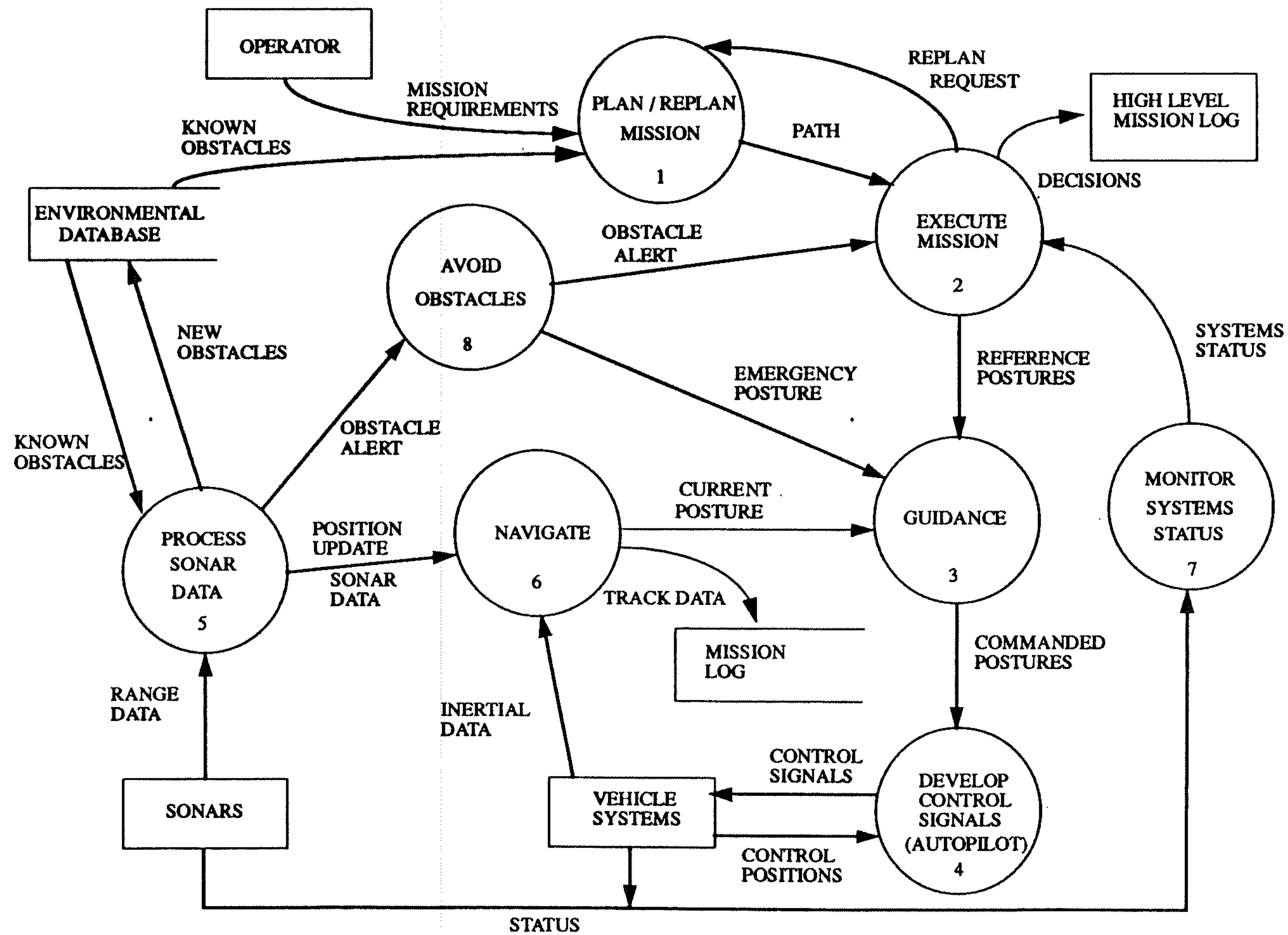


Figure 1. AUV System Dataflow Diagram

are contained in the rule base, which is partitioned into divisions of vehicle maneuvering rules, system monitor rules, navigation rules, environmental hazard rules and specialized mission rules. Input to the Mission Executor consists in both the internal vehicle configuration and mission plans (designed as vehicle postures of nineteen variables at the various waypoints). Output from the Mission Executor consists of final reference postures passed directly to the lower level Guidance system. Lower level Guidance reacts by controlling the autopilot at the next level. Figure 2 shows this decision process.

Input mission postures are first given to a Mission Interpreter which places a posture into the proper object format and designates the high-level classification of the configuration as transit or specialized mission. It further determines a lower level of configuration as a turn, ascent, dive or surfacing based on the succeeding posture. Navigation rules determine whether the next waypoint will be made on time. Obstacles or elapsed time may determine that a new or updated waypoint be constructed. The Navigator Module (external to the Mission Executor) is invoked by SKIPPER for this purpose. In the event that an obstacle or obstacles force a detour in the path execution of the AUV, an Obstacle Avoidance DecisionMaker invokes the replanner (also external to the Mission Executor) to plan a new route to the goal mission area. The new route is evaluated for both proximity to the old route and ability of the AUV to reach the destination and carry out the mission with available battery power.

Measures of uncertainty are used for initial sonar obstacle determinations which SKIPPER receives from the Obstacle Avoidance Decision Maker. As the classification improves, certainty of the obstacle's location better fixes the progress of the transit or mission. It also allows for determination of whether the obstacle(s) in question requires avoidance maneuvers.

IMPLEMENTATION IN CLIPS 5.0

The decision to build a Mission Executor in CLIPS was made in the fall of 1990 based on the rapid prototyping capability of CLIPS. Its LISP-like rules, relative compactness and low-cost are attractive features for a control system designed to fit in a compact real-time testbed. Further strengthening the argument for CLIPS is an evaluation by William Mettrey of Bell-Northern Research which compared CLIPS against other rule-based tools. CLIPS outperformed three of the other four tools (all commercial) (Mettrey 1991). Balanced with its low-cost, it was clearly the winner.

Initial development actually focussed on modeling the internal world of vehicle systems. The model of this ***internal world*** turned out much like the model Giarratano used for his Joe's Object Oriented Database (JOD) (Giarratano 1991a). The implementation is somewhat different. This is not meant to be a user-interfaced advisory system. Using low salience , a monitor-health-continuously rule checks the state of thirteen instances of various equipment objects (nearly as a background function). The equipment objects all share the common attributes that they are being monitored for their respective high/low redline thresholds and high/low guardline thresholds. The system monitor class is further broken down into a sonar class (there are four sonars on the testbed), a control system class, an onboard computer class, a navigation instruments class with instances of dead-reckoning analyzer (DRA) and Global Positioning Satellite (GPS) receiver, and an environmental sensors class [figure 3]. Through queries and daemons, the changing object states cause pattern matches in the system monitor rules.

Decision-making, while contained in the rule base, is preserved in the object base by the

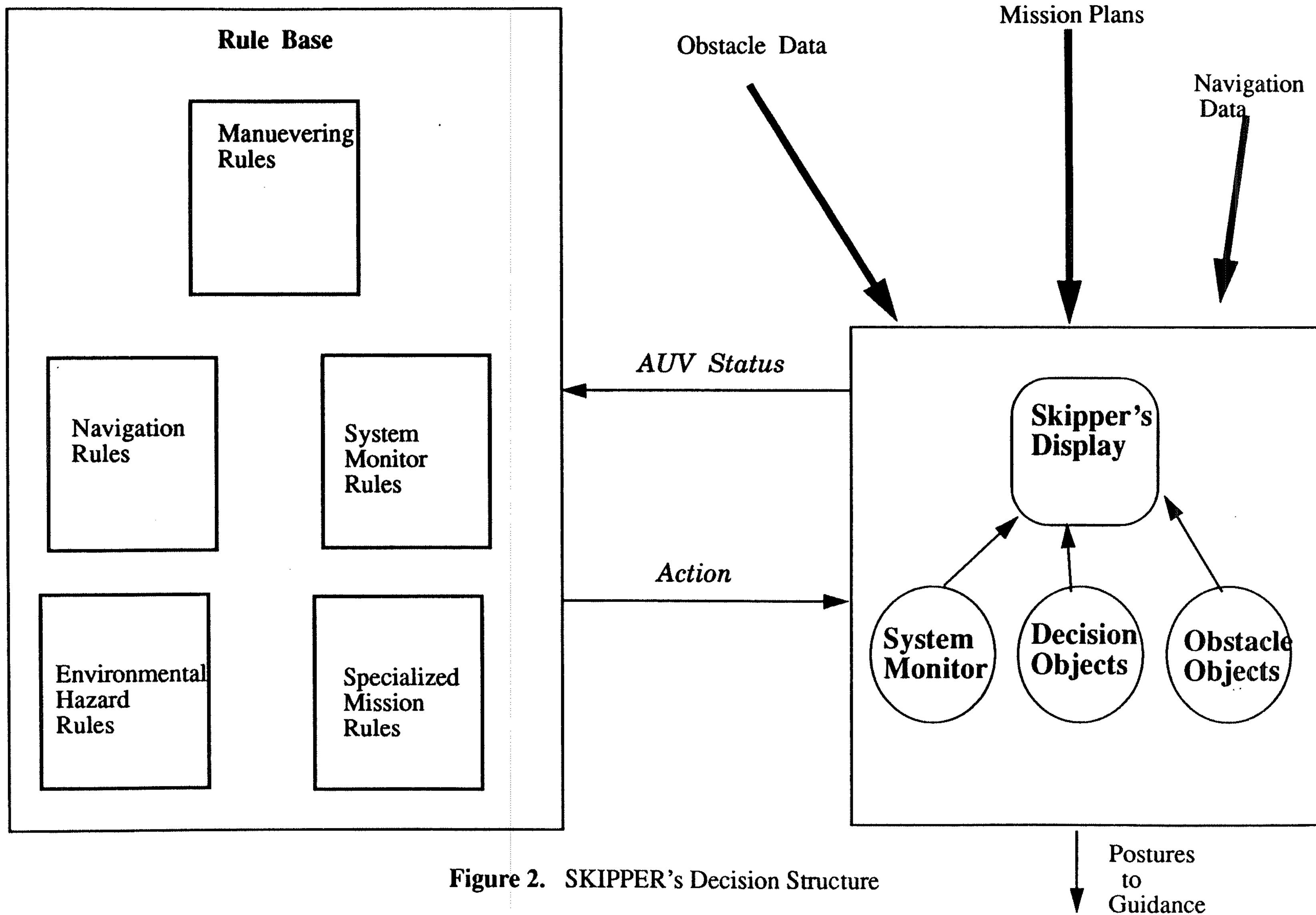


Figure 2. SKIPPER's Decision Structure

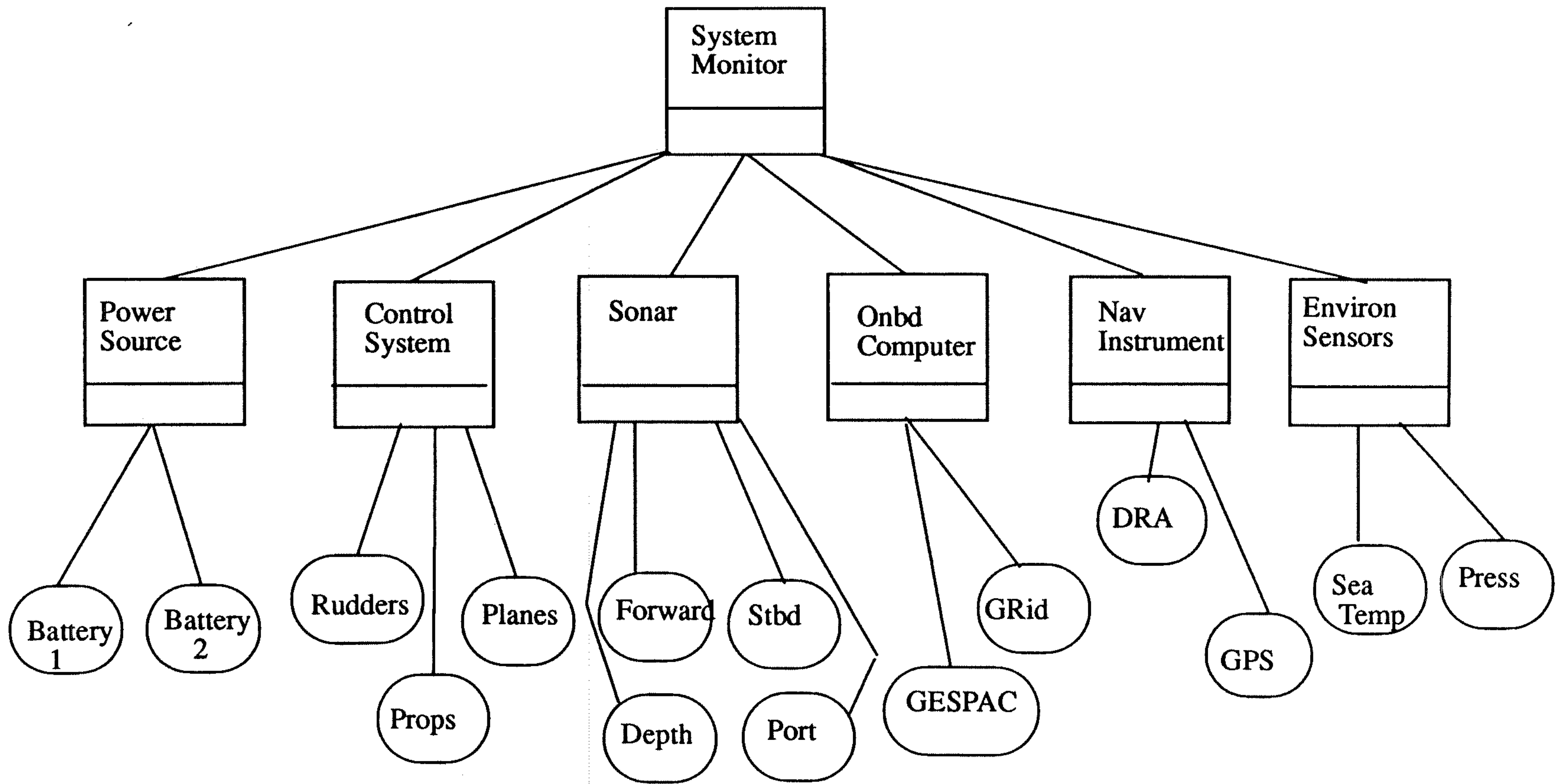


Figure 3. System Monitor Objects

decision-objects. This is because some decisions may require knowledge of previous decisions. This is particularly true for the high-level mission decisions. The design of the decision objects incorporates slots for high-level mission decisions, lower-level manuever decisions, navigation decisions, system-monitor decisions, and special-mission decisions. In addition, provision is made for time-stamping the decision. Literally any decision-change will cause a new decision-object to be created, as a record must be maintained of all decisions. The instance **current** is copied to another unnamed instance using *deffunction* calls, as shown in the following example:

```
(deffunction copy-old-instance (?instance)
  (send (symbol-to-instance-name ?instance) put-mission_decision
     (send [current] get-mission_decision))
  (send (symbol-to-instance-name ?instance) put-maneuver-decision
     (send [current] get-maneuver_decision))
  (send (symbol-to-instance-name ?instance) put-sysmonitor_decision
     (send [current] get-sysmonitor-decision))
  (send (symbol-to-instance-name ?instance) put-navigation_decision
      (send [current] get-navigation-decision))
  (send (symbol-to-instance-name ?instance) put-special-mission-decision
      (send [current] get-special-mission-decision))
  (send (symbol-to-instance-name ?instance) put-justification
      (send [current] get-justification))
  (send (symbol-to-instance-name ?instance) put-decision_time)
     (send [current] get-decision_time)))


(defclass DECISION (is-a USER)
  (slot mission_decision (multiple))
  (slot maneuver_decision)
  (slot sysmonitor_decision)
  (slot navigation_decision)
  (slot special_mission_decision)
  (slot justification)
  (slot decision_time))


(deffunction maneuver-decision-change-obstacles (?change ?justification)
  (bind ?name (gensym*))
  (make-instance ?name of DECISION)
  (copy-old-instance ?name)
  (send [current] put-maneuver-decision ?change)
  (send [current] put-justification ?justification)
  (send [current] put-decision_time (time)))
```

Certain physical changes to the vehicle's environmental or internal world may improve the state

of the vehicle somewhat. Yet, the mission-world must dominate behavior. If a mission decision was previously made to *continue_with_restrictions or abort_mission*, improvement in the other two worlds may or may not justify improvement to *continue unrestricted.* To prevent a collision of *defrules,* another basis must be used, such as the justification for the continue with restrictions state. Retrieval of the justification for the previous mission status may involve searching back over several state changes. This should not involve a lengthy amount of traversal. This is more easily done with hyperlinks between objects or a simple query rather than a linked-list. The following example of a post-casualty vehicle recovery rule highlights this. While the left-hand side (LHS) conditions indicate that the mission may be fully recoverable, the right-hand side query hunts for the existence of the only possible justification for full recovery. This further requires a call to a deffunction to determine if the mission is physically recoverable in terms of mission parameters mission-critical power and distance/time-to-go (called from the navigation module external to the Mission Executor).

```
(deffunction recovery-mission-evaluation ( ?location )
  (if (or (< (send [battery] get-power_status) ?*mission-critical-power*)
        (> (navigator-update-from ?location)  ?*recovery-time*)) then
    (send [current] put-mission-decision Abort_Mission)
    (send [current] put-justification mission-deviation-nonrecoverable)
   else
    (send [current] put-mission-decision Continue_Unrestricted)
    (send [current] put-justification mission-deviation-recoverable)))
```

```
(defrule vehicle-recovery-state
  (mission_status Continue_with_Restrictions)
  (system-monitors normal)
  (location ?location)
  (or (redundant_system_online ?system)
      (normally-operating ?system))
 =>
  (do-for-instance ((?ins DECISION)) (eq mission-deviation
                  (send ?ins get-justification))
  (recovery-mission-evaluation ?location )))
```

Certain high-level behaviors, such as the overall mission decision are modeled using the Artificial Neural Paradigm implementation suggested by Giarratano (Giarratano 1991). This application of salience is useful in differentiating between a high-level, less frequent macro-action and a lower-level frequently performed action. The philosophy for using salience in this manner is that a situation (pattern match) which may cause a mission abort usually requires immediate or timely reaction and certainly takes precedence over a routine action such as a normal turn or depth change in a deep-water open-ocean environment. The emergency-action rule must be guaranteed firing before other semantically lower-priority rules on the *agenda.* This (however loosely) heuristically models a submarine commander's "situational awareness" in an emergency

```
(defrule emergency -evasive-maneuver
  (declare (salience 1000))
  (obstacle-proximity  ?direction  danger-close)
  (maneuver-available ?maneuver)
  (system-monitors  ?status)
  (not (previous_mission_decision abort-mission))

  =>

 (assert  (emergency-guidance ?maneuver))
  (assert  (mission-decision  alter-track))
  (Replanner get-new-route  ?position))



 (defrule battery-power-guardline
  (declare  (salience  ?*sysmonitor-salience*))
  (mission-percentage ?percent&:(< ?percent  70))
  (battery ?number  at-guardline)
  =>
  (bind ?*sysmonitor-salience* (+  ?*sys-monitor
    salience   100))
  (assert   (mission-status critical)) )
```

Figure 4. Setting Precedence with Salience in SKIPPER

[figure 4].

Salience is also used in some background functions such as the sequencing of the mission timer and the loop which causes the slots of the respective system monitors to be queried on a nearly continuous basis. Still, it is used sparingly. SKIPPER still retains a strong declarative nature. The rest of the rule base pattern-matches on the objects are of normal undeclared salience.

CONCLUSION

Successful software for an AUV must incorporate techniques from artificial intelligence, real-time processing, environmental sensing, and vehicle maneuverability into a compact integrated package. This is due to an AUV's lack of human control during mission execution and the inability for human intervention in the event of unforeseen problems. In addition, many tasks are knowledge-intensive and require domain-specific information. Therefore, the ability to include autonomous intelligent decision-making on an AUV is essential for its satisfactory performance. With the accumulated experience in submarine operation, we believe many of the onboard problem-solving and reasoning can be adequately modeled using a rule-based system. The Mission Executor is designed to (1) monitor relevant vehicle variables, component parameters, and environment data; (2) ensure the progress of pre-planned mission execution; and (3) in the event of unplanned interruptions during a mission, be able to diagnose the problematic situations and enable the vehicle to adapt to the unexpected environment by manipulating and changing vehicle and mission parameters.

A prototype for the Mission Executor has been completed and will be incorporated in the testbed as dependent modules are finished. The design is one that is extensible. Further, its object-oriented nature allows for incremental construction and testing of modules in relative isolation. The specific mission modules are areas for more fine-grained research. Because of the specialized nature of each of the mission modules, they are excellent areas for application of object-oriented tools like CLIPS 5. 0.

ACKNOWLEDGMENT

This paper was prepared in conjunction with research funded by the Naval Postgraduate School.

REFERENCES

Bellingham, J. G., T. R. Consi, R. M. Beaton and W. Hall (1990). Keeping Layered Control Simple, *Proceedings of IEEE Symposium on Autonomous Underwater Vehicle Technology*, Washington, DC, June 1990.

Blidberg, D. R., S. Chappell, J. Jalbert, R. Turner, G. Sedor, P. Eaton (1990). The EAVE AUV Program at the Marine Systems Engineering Laboratory, *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October, 1990.

Brooks, R. A.(1986). A Layered Intelligent Control System for a Mobile Robot, in Gangeras and

Girald (eds), *Robotics Research*, MIT Press, Boston.

Cloutier, M. J. (1990). *Guidance and Control System for an Autonomous Vehicle*, M.S. Thesis, Naval Postgraduate School, June 1990.

Giarratano, J. C. (1991). *CLIPS User's Guide Volume 1: Rules, CLIPS Version 5.0*, NASA-Lyndon B. Johnson Space Center Information Systems Directorate Software Technology Branch, January 1991.

Giarratano, J. C. (1991a). *CLIPS User's Guide Volume 2: CLIPS Object Oriented Language*, NASA-Lyndon B. Johnson Space Center Information Systems Directorate Software Technology Branch, April 1991.

Healy, A. J. , R. B. McGhee, R. Cristi, F. A. Papoulias, S. H. Kwak, Y. Kanayama and Y. Lee (1990). *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October, 1990.

Lee, Y., Luqi and R. B. McGhee (1991). Automating the Construction of Real-Time Software for an Autonomous Underwater Vehicle through Prototyping, *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology,* Durham, New Hampshire, 23-25 September 1991.

Mettrey, W. (1991). A Comparative Evaluation of Expert System Tools. *IEEE Computer*, Vol. 24, No. 2, February 1991, pp. 19-31.

Zheng, X., E. Jackson, and M. Kao (1990). Object-Oriented Software Architecture for Mission-Configurable Robots, *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments,* Monterey, California, October, 1990.