SOLON: An Autonomous Vehicle Mission Planner

M.J. Dudziak
Martin Marietta Baltimore Aerospace
Baltimore, MD 21220

Presented at the Workshop on Space Telerobotics,
Jet Propulsion Laboratory, Jan. 20-22, 1987

S30-.60
120
188112

I) Introduction:

The SOLON Planner provides an architecture for effective real-time planning and replanning for an autonomous vehicle. The acronym, SOLON, stands for: State-Operator LOgic MachiNe; the highlights of the system, which distinguish it from other AI-based planners that have been designed previously, are its hybrid application of a state-driven control architecture and the use of both schematic representations and logic programming for the management of its knowledge base (1).

SOLON is designed to provide multiple levels of planning for a single autonomous vehicle which is supplied with a skeletal, partially-specified mission plan at the outset of the vehicle's operations. This mission plan consists of a set of objectives, each of which will be decomposable by the planner into tasks. These tasks are themselves comparatively complex sets of actions which are executable by a conventional real-time control system which does not perform planning but which is capable of making adjustments or modifications to the provided tasks according to constraints and tolerances provided by the Planner.

The current implementation of the SOLON is in the form of a real-time simulation of the Planner module of an Intelligent Vehicle Controller (IVC) on-board an autonomous underwater vehicle (AUV). The simulation is embedded within a larger simulator environment known as ICDS (Intelligent Controller Development System) operating on a Symbolics 3645/75 computer (2).

II) Real-Time Operational Context

Many planning systems have been developed over the years of AI research, but few have been built expressly for use in the control of an autonomous vehicle which will be operated in extremely remote, inaccessible environments (3). Most autonomous vehicles to date have been land-based,and the focus of research has been upon problems of mobility and obstacle avoidance. We recognize that the AUV problem includes all of the same issues faced in the design of any autonomous vehicle but in our case long-range goals dictate the need for modularity and interchangeability of system components (e.g., sensor subsystems, manipulators, photographic equipment, etc.) and the integration of these subsystems with the controller responsible for moving the actual vehicle. Furthermore, an onboard IVC must be capable of performing in real time, without the typical options that are generally available to terrestial vehicles. An AUV has additional directional degrees of freedom than a terrestial vehicle as it moves through 3-space but it has less operational freedom to stop or to undo certain motions. For example, with the exception of certain experimental-only vehicle architectures, an AUV cannot retrace its path exactly and bringing the vehicle to a halt cannot make use of braking systems or high degrees of friction.

The underlying philosophy of the SOLON planner includes the following basic premisses:

i) Both the knowledge base and the work (the processing) must be distributed, allowing for concurrent processing by different components, in order to solve the real-time bottleneck;

ii) The architecture must be dynamic and flexible to allow for the fact that different environmental and decision-making situations will require a different balancing of the processing loads;

iii) Any intelligent controller must evolve in its design from the simple to the complex

and that vehicle-specific and mission-specific requirements which are not presently foreseeable should not require major restructuring of a system design or implemented, operable software.

III) High-Level Architecture

SOLON is composed of five logical processes (logprocs) which operate as cooperative but autonomous agents in the Planner. These are:

i) *Strategist* - central decision-making; selection and scheduling and detailing of objectives and tasks within objectives;

ii) *Event Assessor* - processing of events and conditions reported by other IVC modules about outside environment or internal vehicle systems;

iii) *Tactician* - interfacing between Planner and lower-level vehicle control systems responsible for driving actuators and effectors;

iv) *Plan Simulator* - evaluating alternative plans in cases where Strategist's first-cut logic cannot derive a clear and distinct best choice;

v) *Plan Registrar* - central editing and modification of the Active Plan.

Each logproc communicates with others by means of an message-passing system which is also the structure employed for inter-modular communications within the whole IVC. The logprocs may be multiple processes which share a single-processor machine such as the Symbolics or they may be implemented in a distributed processing environment where each logproc has its own physical processor. Their relationships are described in a data flow diagram *(Figure 1)* and their functions are described in more detail later in this paper.

IV) Distribution of Intelligence and Decision-Making

A major stumbling block to the implementation of real-time AI systems has been in the area of performance. Typically a system is bound with achieving a particular goal and events occur which cause one of the following types of situations:

i) The original problem that the system is working upon is no longer relevant

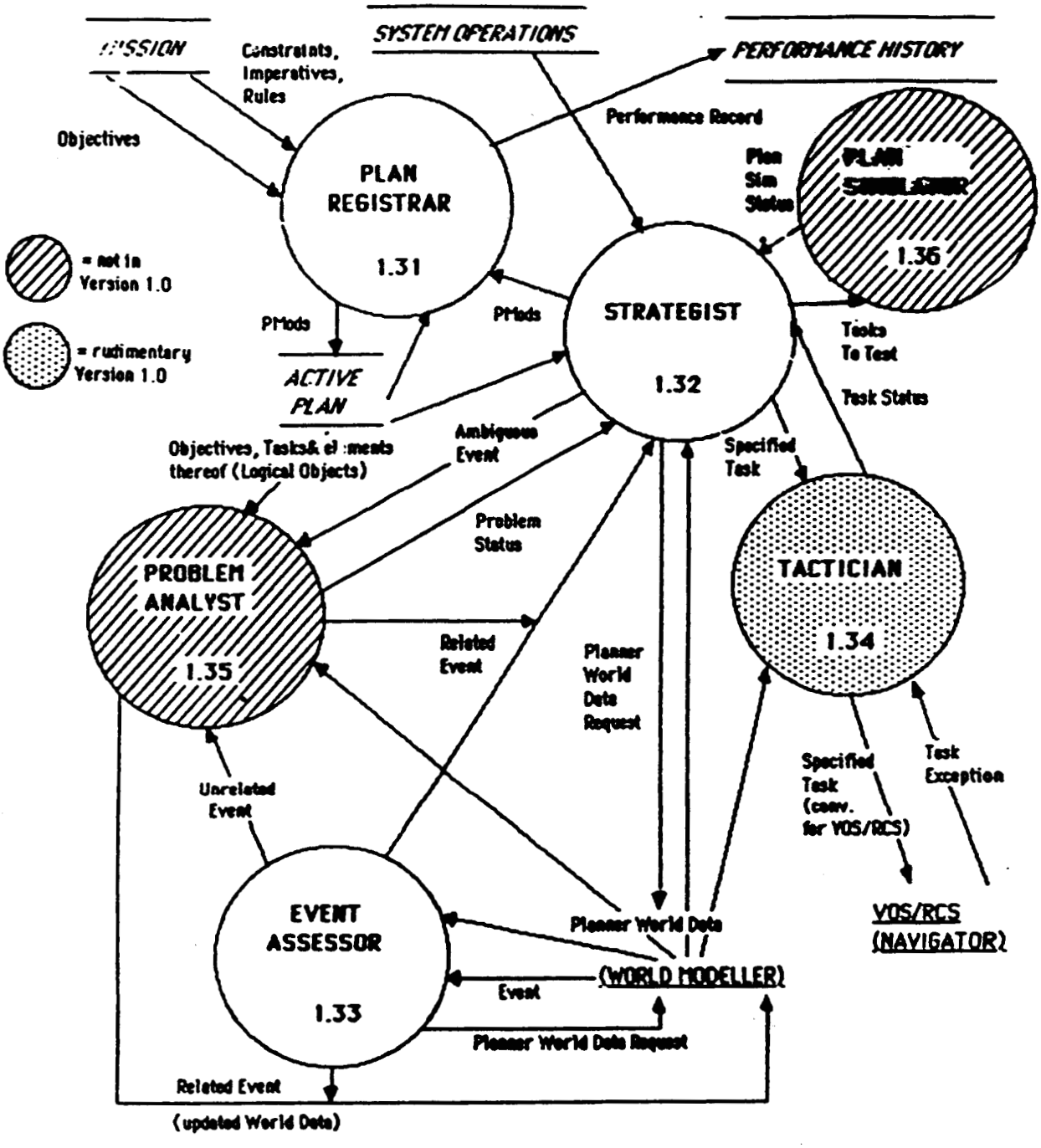
ii) The data being used by the system in its current problem-solving has been changed

iii) A new problem has higher priority and should take precedence over the current activities of the system.

This is more than a problem of conflicting goals in which the system must decide to satisfy one. Rather, we are faced with goal conflicts that may not arise until after a given attempt to satisfy a goal is underway and which may not be communicated to the system to satisfy the goal. The cost in time and computation resources for determining how to alter the current activity and how to respond to a new situation may be prohibitive given the other requirements of the control system.

The approach undertaken in SOLON distributes the work among the multiple logical processes which can proceed independently. The performance of the Plan Simulator is not impacted by the Strategist, which may be in an idle state or else busy running PROLOG inferences, other than by the constraints imposed by the scheduler if both logprocs are running off the same physical processor. Obviously there is signficant advantage to a logproc having its own processor. At any given time, messages can be sent between logprocs in order to interrupt or provide additional information. There are a finite number of message types (potential-obstacle, task-completed, relevant-object, task-status, etc.) which may be

FIGURE 1 - High-Level Planner Architecture



sent between logical processes but the schema-based representation structure allows for this number to grow as the complexity of the Planner implementation increases.

When a message is sent, for instance, from the Tactician to the Strategist, the message is sent with the priority that has been assigned by the sender. This indicates how significant or urgent the sender believes the message to be. That will not necessarily be the way the message is treated by the receiver, which may be working on a task that is the result of a previous message and which may be rated with a much higher priority.

When the receiver gets a message, it is not automatically interrupted from its work. Instead, each process follows a cycle of execution followed by checking for messages. This has been implemented for the Strategist and is described further in section X. If the receiver is at a point where it can be interrupted, then it will determine if there is any message in its queue that demands a jump to a new task. The receiver process uses the Process-Msg function to determine what message is the most critical one in the queue. Currently Process-Msg is a LISP function that employs several simple rules but in future implementations it will be a compact expert system in its own right within the Planner.

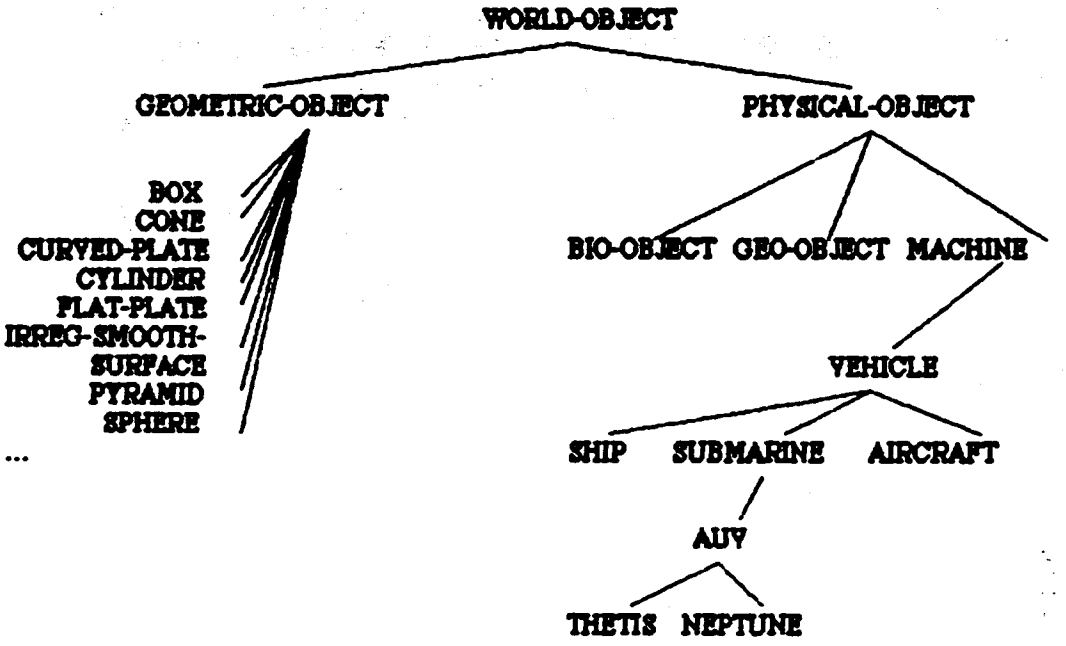
What becomes of the work that a logproc was doing if and when it is redirected by a message to tackle some other task? There are two possible approaches. One is to interrupt the original task, save a history of what was going on and possibly resume it at a later time. However, this poses many problems in truth maintenance and non-monotonic reasoning because the situation that gave rise to the original task may be altered as a result of the new operations being executed. Rather than attempt to keep track of all changes and determine that the original task should be resumed or not, the approach in SOLON is to simply drop the original task altogether. If in fact the original task should be resumed, then the conditions which led to that task will surface again in the form of a message from one logproc to another. Otherwise the system will deal with the highest priority messages that currently exist. The Planner is always responding to the latest, most current state of affairs, even if this means doing some extra work or duplicating a few steps to determine what should be done next. In other words, one may ask the question "What is the most pressing business right now?" more often, but one will always have the best and most up-to-date answer to that question, given the rules that have been built into the Planner through its functions and specific PROLOG axioms.

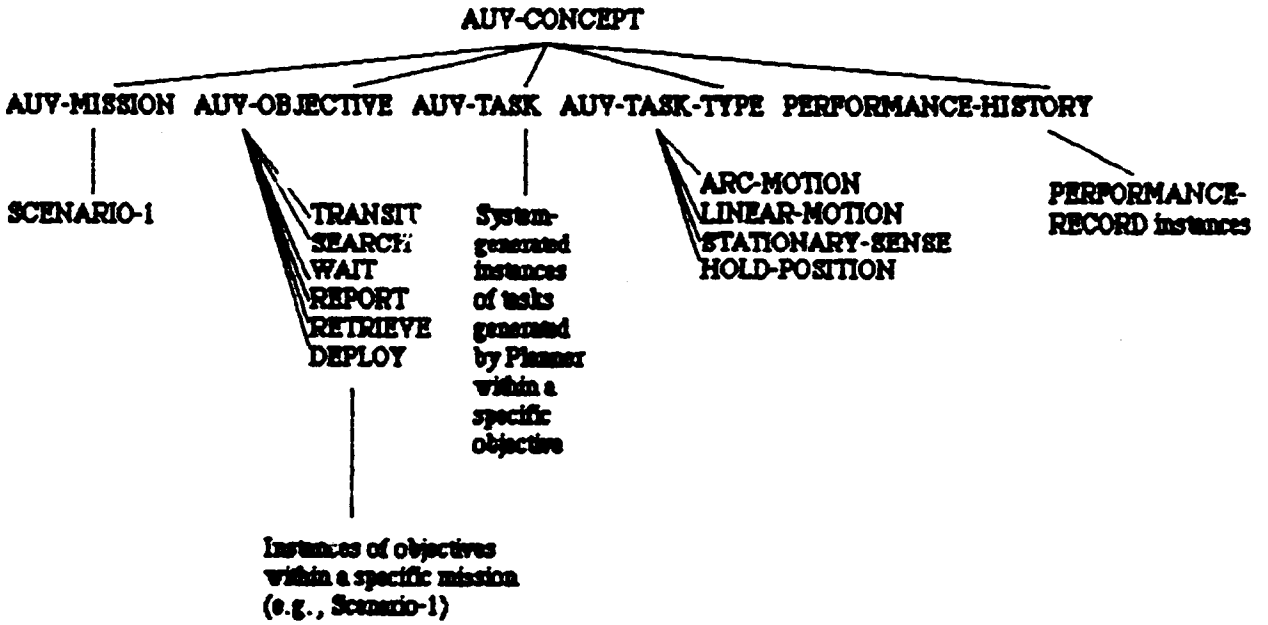
VI) Multiple Forms of Knowledge Representation

Factual knowledge of the environment and the vehicle is maintained in a database which employs the schema-slot-value structure as implemented in Knowledge Craft, a Common LISP based system tool developed by Carnegie Group, Inc. There is a fundamental taxonomy of world objects and relations which hold between various objects, as indicated in *Figure* 2. Generally, knowledge about "what is" is stored in this schematic representation, whereas "how to" knowledge, rules and operational guidelines, is represented in PROLOG axiomatic expressions. The PROLOG mechanism was deemed to be suitable for making queries and deriving solutions that could best be found using backward-chaining inference processes. However the PROLOG form was not deemed sufficient to be used as the only form of knowledge representation.

The power of the schematic representation is that the facts are easily stored in a well-structured format and the information is easily accessible. Moreover, from a

FIGURE 2 - Fundamental Objects and Relations in the SOLON World





experimentalist point of view it is a good representation for building a prototypical system, one that is open-ended and intended to evolve beyond one's current expectations.

Vehicle missions consist of human-specified skeletal plans, the elements of which are known as objectives - high-level types of operations such as transiting to a destination, searching a region for a given set of objects, following a moving object, etc. Objectives are deocmposable into tasks, more elementary types of activity (transit to waypoint, hold position, send transmission, manipulate arm or sensor platform, etc.). Mission specifications are, therefore, short program-like structures that provide the Planner with a first-cut set of objectives to perform in a given temporal sequence with some parameters of those objectives detailed and others left for the Planner to address at run-time. The sequence of objectives and the parameters (e.g., bounds of a region to search, starting position for a search operation) are the default or first-choice values for the Planner to use as optimal guidelines. Through the built-in logic of the Planner and rules that are specified within the mission (see next section) the sequence and parameters and the choice of tasks for satisfying each objective are modified in response to events that occur both in the external environment and within the vehicle.

VII) Hierarchy of Operational Rules

A considerable database of operational rules is implemented using PROLOG. Certain rules are applicable to any mission for any type vehicle and these may be considered the most basic planning rules, modification of which constitutes a redesign of the planning algorithms. These rules are stored in a structure called Act-Prolog and are always present in the working set of PROLOG axioms available for inferencing operations. Other rules are specific to the vehicle but apply to any mission that such a vehicle might undertake. They are known as Sys-Oper rules and will also be available during the entirety of a given mission but may be replaced at pre-deployment time without disturbing the Act-Prolog body of rules. Obviously, this type of partitioning is principally for the benefit of easy system maintenance and modification and does not affect the actual Planner operations.

A similar body of rules are global for a given mission and are accessible at all times during the Planner operations for any objective that is a component of the mission. Next there are three classes of rules which are not accessible uniformly; these include:

i) rules that apply during a specific type of mission objective; i.e., a special rule for search operations, which does not apply to transit or escort operations;

ii) rules that apply for a specific type of task within an objective; i.e., a special rule that applies to 'waiting' tasks and does not apply to motion-oriented tasks;

iii) rules that apply for a specific objective within a mission; i.e., a special rule for when the vehicle is searching area B for a sunken object and only for that objective, having no applicability to other objectives within that mission.

The purpose of this subdivision of rules is to allow efficient entry of new rules into the system and to manage the PROLOG inferencing process so that the system never has to deal with more than those rules which could possibly apply at a given instance. The aim is to reduce the size of the rulebase wherever possible, reducing the number of rules that must be examined and eliminating from consideration those rules that could not possibly have any relevance at a given instance.

VIII) The Strategist

The Strategist is clearly the most complex logical process within the Planner, and it is within the Strategist that the state-operator architecture is employed most fully. It is also the logical process that makes most extensive use of the message processing and evaluation logic which is a critical element in the SOLON design. Messages which arrive from other Planner logprocs are processed according to an algorithm which attends to both the sender's and the receiver's evaluation of the message priority and significance for the current state of the vehicle's mission plan. The algorithm we have initially employed is a simplified version of what we expect will evolve into an expert system in its own right, a clear place-holder for future machine learning studies. Depending on the message chosen to address, the Strategist moves into one of several state-operator functions or processes (currently all are implemented as LISP functions) and future actions of the Strategist are then governed by two factors:

i) The results of function evaluations and hypothesis generations within the current state-operator;

ii) The appearance in the Strategist's message-queue of a "critical message" demanding immediate attention and the overriding of current replanning activities.

Examples of current state-operators include:

i) Analyze-Next-Scheduled-Objective

ii) Expand-Objective-Into-Tasks

iii) Examine-Task-Status-Msg

iv) Determine-Plan-Change-Directive

Within each state-operator, the main activities consist of determining what is the appropriate hypothesis to test and then making queries into the PROLOG-based knowledge base. This consists of a dynamic set of axioms (modifiable facts and constant rules). Through the PROLOG mechanism employed within our implementation (CRL-PROLOG) these have full access to the major body of represented knowledge about the vehicle, mission plan and environment, which is in the form of schema-slot-value-relation data structures.

The basic algorithm of the Strategist top-level function is presented in *Figure 3*.

FIGURE 3 - Strategist Top-Level Algorithm

```
[Outer loop]
    IF (Critical Message Received)
    THEN (Make the Critical Message the Current Message)
    ELSE (Get the Most Relevant Message From the Queue if there is one)

    Based on the type of message [related-event, obstacle, task-status, etc.]
    received, (Select Appropriate State-Operator To Activate)

    IF (Strategist is in wait-state)
    THEN (Put Strategist process into wait state)
    ELSE
        [Inner loop]
            UNLESS (Strategist is directed into wait-state)
                   OR (Critical Message Received)
            (Execute the Selected State-Operator)
```

IX) A Working Example

The operation of the Strategist may best be described through the use of an example scenario which will illustrate the manner in which the state-operator functions and PROLOG rules operate to control the vehicle and replan its mission. We will consider a simple search mission as described by *Figure 4*. The tentative path of the AUV, were it to execute each of the initial scheduled objectives in sequence and without any replanning due to new events, is indicated by *Figure 5*.

Initially the Strategist is given a start-mission message which activates the state-operator Analyze-Next-Scheduled-Objective. The obvious first step is to consider what is next on the list of scheduled objectives given by Mission Control. A series of Prolog queries are generated to determine if there are sufficient reasons for undertaking this objective and no overriding reasons to avoid this objective at this time.

Assume that the first objective is to move the vehicle to a given point A. This objective is represented as Transit-A and it inherits all the default characteristics associated with transit operations, as well as any special rules that may have been specified by Mission Control in the skeletal plan. These defaults include the decomposition of the objective into component tasks. For point-to-point transits, there is only one elementary task, that of moving the vehicle in 3-space. However, before asserting a new transit-type current-task, checks are made to determine that there are no known obstacles (which may have been detected by the sensors) in the vehicle's path. Using an algorithm which treats potential obstacles as expanded spheres and represents the vehicle as a point, lines of tangency are computed which provide possible new waypoints for the vehicle to use in navigating around the obstacles. An elementary set of rules determines which is the best set of alternative paths and these paths become the new component tasks for the current objective.

Suppose that the vehicle is now engaged in a search objective, where there are a specified number of objects (cylinders or curved surfaces with a radius > 3m but < 10m) which are deemed relevant to the search and one particular object (a pipeline on the seafloor) which is the goal object for the search. Specific rules provided with the mission plan indicate the actions to be taken if and when various relevant objects are detected. These are in the database of rules which are active while the vehicle is executing its search objective. Having received a message about a cylindrical object at point A from the World Modeller, the Event Assessor determines that the object is relevant to the current objective and the Strategist is notified. The latter responds by determining an appropriate change that applies to the current state of the Active Plan. This response, based upon rules input with the mission specification, may be to initiate a new objective, a spiral search operation centered upon the newly discovered object. Before actually changing activities, the Strategist invokes a state-operator which explores "what if" type queries to determine if instituting the change of plan would cause conflicts for other objectives of the mission. If there are significant conflicts, then the Strategist returns to the state-operator charged with determining an appropriate change of plan given the new event (object). Otherwise, the new objective is treated as the next scheduled objective (as if it were part of the original schedule) and the state-operator charged with checking out all next-scheduled objectives is then activated.

FIGURE 4 - Search Mission Scenario

SCENARIO-1 Test Mission--

TRANSIT-A (move vehicle in straightest possible path to point (200 200 200))

LADDER-SEARCH-A (search 200m x 200m region for sunken submarine vessel, using ladder-like motion pattern; relevant objects include smooth, curved surfaces and metallic objects)

TRANSIT-B (move from end of search operation to point (1200 1200 400))

LADDER-SEARCH-B (search 400m x 400m region for same submarine)

TRANSIT-C (move to point (2000 800 800))

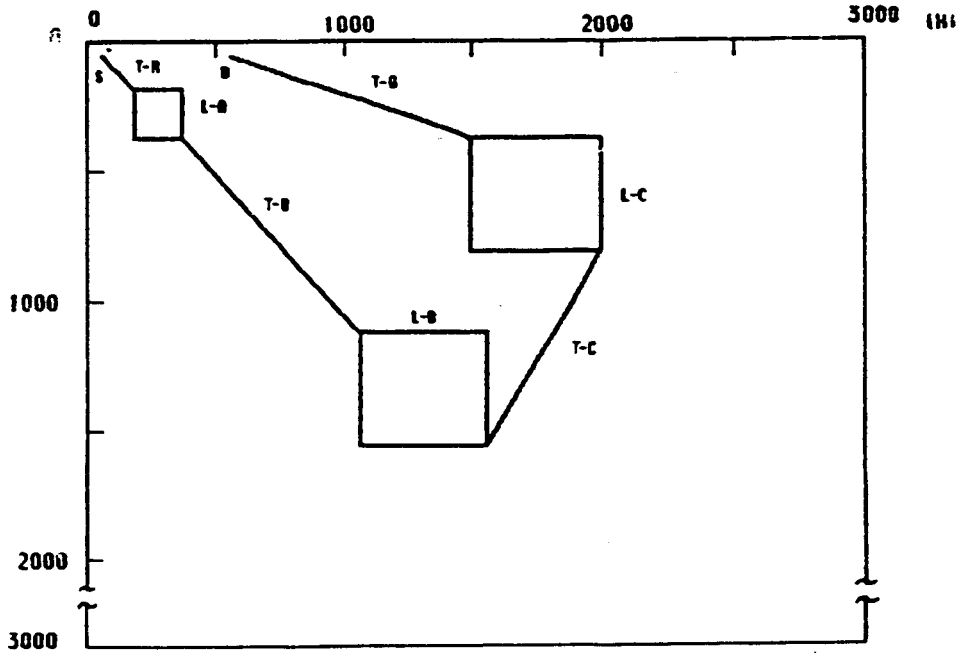
SPIRAL-SEARCH-C (search 600m radius region starting from current position, moving counter-clockwise in a spiral motion pattern)

TRANSIT-D (move to point (400 60 0))

REPORT-TO-BASE (transmit data from surface using radio)

FIGURE 5 - AUV Scenario Path

The world is a 3 km by 3 km square of coastal ocean of varying depths.



The mission is to search for a sunken submarine in this hypothetical ocean space.

The initial mission specification given to the AUU consists of eight objectives: Transit-A (go to pt.A), Ladder-Search-A (search an area using a ladder-like motion pattern), Transit-B, Ladder-Search-B, Transit-C, Ladder-Search-C, Transit-D and Report-To-Base.

When the next-scheduled objective "checks out" as being satisfiable and not in conflict with other objectives (according to the rules provided for determining conflict and satisfiability), a state-operator is activated which "expands" the objective into appropriate component tasks. In the case of transiting operations, these tasks will generally all be motion-oriented, but in the case of a search objective there may be an interspersing of different types of tasks among the motion segments required to move the vehicle around. These include performing intensive scanning of a region while the vehicle is at an intermediary waypoint in the search, manipulating objects if the vehicle is equipped with appropriate devices, and so forth. Once the new current objective has been "expanded" into an initial set of tasks, these tasks are sequentially processed in a similar fashion - the next scheduled task is "checked out" just prior to execution for consistency with the actual state of affairs (which may have changed significantly since the objective was expanded into a list of tasks) and if the new task is approved, it is then transmitted via the Tactician to the Vehicle Operating System, a control program whose function is, akin to the mechanical engineering staff on a ship, to carry out the high-level commands and operate the servos and actuators of the vehicle. Communications from the Vehicle Operating System back to the Planner consist of messages indicating the status of the given task - either that it has been completed or that it cannot be completed, given the constraints specified (e.g., maximum time to perform a transit, maximum deviation from a given course).

Fundamental to the SOLON architecture is this partitioning or distribution of jobs among many different agents, the state-operator functions (4) of the Strategist. Many independent specialists, as it were, handle their particular tasks, without burdening each other or the higher-level modules in the Planner. However, with the message-passing system that runs throughout SOLON, the higher-level modules, like the captain and officers of a ship, have access to the activities of the lower-level units and are able to make changes which can include changing the tasks of those lower, simpler units.

X) Conclusion

The SOLON Planner provides several new features which we believe are important for planning and reasoning, particularly in a real-time mobile context. First, it provides a mechanism for distributing or partitioning the knowledge required for high-to-medium-level vehicle control into a number of different representation schemes (rather than just one method) and into a number of independent but communicating databases. Secondly, SOLON provides a mechanism for distributing the work of evaluating alternatives and selecting sequences of objectives and tasks among several agents (logprocs) which can readily be implemented in a concurrent, MIMD-type machine architecture. Thirdly, SOLON operates by "default reasoning" principles - the network of state-operators is such that problem-solving is attempted first using the simplest, most probable or most expected searches and queries. When the default methods fail, more complex logic is invoked. There is a definite redundancy built into SOLON; certain logprocs will receive messages and initiate activity which may turn out to be not required because of solutions implemented by another logproc. (This is particularly true in some cases of obstacle avoidance). This redundancy, we feel, is not only admissible but important. The justification for such mechanisms and for a good part of the

state-operator architecture lies not only with the failure of many previous planning systems but the obvious historical success of biological planning in humans and animals.

(1) Foundations and influences for our research include work which is described in the following sources:

Brooks, Rodney, "Long-Term Autonomy for Mobile Robots", *1986 MIT Sea Grant College Program Lecture and Seminar Series*, MIT, Cambridge MA, October 1986

Durrant-Whyte, Hugh F., "Distributed Coordination and Decision Making in a Robot Sensing-Action System", PhD. dissertation, Univ. of Pa., Philadelphia PA (private circulation)

Guha, A. K. & Dudziak, M. J., "Knowledge-Based Controllers for Autonomous Systems", *Proceedings of the 1985 IEEE International Symposium on Intelligent Control*, Troy NY, August 1985

Hayes-Roth, B., Hayes-Roth, F., Rosenschein, S. & Cammarata, S., "Modeling Planning as an Incremental, Opportunistic Process", RAND Corporation, Santa Monica CA, June 1979

Shank, Roger & Abelson, R. P., *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum, Hillsdale NJ, 1977

Sowa, J. F., *Conceptual Structures*, Addison-Wesley Systems Programming Series, Reading MA, 1984

Wesson, R. & Hayes-Roth, F., "Dynamic Planning: Searching Through Space and Time", RAND Corporation, Santa Monica CA, (Report P-6266)

(2) The overall IVC and the ICDS simulator are described in more detail within a related paper: Dudziak, M., Hall, M. & Zaret, D., "IVC : An Intelligent Vehicle Controller with Real-Time Strategic Replanning", *Proc. of the IEEE 1987 International Symposium on Intelligent Control*, Phila PA, Jan. 1987

(3) Principal foundations for our approach to autonomous-vehicle-specific problems include:

Harmon, S. K., "Planning for Transition Unknown Natural Terrain", *Proceedings of the DOE CESAR Workshop on Planning and Sensing for Autonomous Navigation*, Los Angeles CA, August 1985

Kanayama, Yutaka & Yuta, Shin-ichi, "Concurrent Programming of Intelligent Robots", *Proceedings of the 8th Int'l. Joint Conf. on Artificial Intelligence*, August 1983

Orlando, Nancy E., "Interfacing Intelligent Software to Robotic Peripherals", *Proceedings of the First International Conference on Applications of Artificial Intelligence to Engineering*

Problems, Southampton Univ., England, April 1986

(4) In our design the state-operators are entities represented using the common schema-based structure employed throughout SOLON. They are program objects which have among their attributes (slots) a list of LISP functions (usually only one) to be evaluated when the state-operator has been activated. These functions can, in a multiple-processor ("parallel machine") environment, easily be converted into individual processes running on their own CPUs. In this way the SOLON system is evolvable in two important respects:

First, new state-operators may be added and integrated into the overall state-operator network structure without disruption to other parts of the net, as it is deemed necessary to subdivide jobs among state-operators or handle new dimensions of the planning problem;

Second, state-operators which are currently functions or processes sharing a single CPU resource may be moved off to other processor machines as the work-load grows to an point where multiple processors become important for maintaining required real-time performance.