

REAL-TIME HIERARCHICAL PLANNING FOR MULTIPLE MOBILE ROBOTS

Martin Herman and James S. Albus

Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

The Multiple Autonomous Underwater Vehicles (MAUV) project is described. The goal of the project is to have multiple underwater vehicles exhibiting intelligent, autonomous, cooperative behavior. The MAUV control system is hierarchically structured and incorporates sensing, world modeling, planning and execution. The levels in the hierarchy include a mission level, a group level, a vehicle task level, and an elemental action level. Issues of real-time planning and dynamic replanning in unstructured environments are discussed.

1. Introduction

The NBS Multiple Autonomous Underwater Vehicles (MAUV) project involves the development of a real-time intelligent control system that performs sensing, world modeling, planning and execution for underwater robot vehicles. The project is funded by the DARPA Naval Technology Office. The goal of the project is to have multiple vehicles exhibiting intelligent, autonomous, cooperative behavior. Our initial tests have involved two identical vehicles engaged in various scenarios in Lake Winnepesaukee in New Hampshire. All software for controlling the vehicles reside on computer boards mounted on-board the vehicles.

This paper focuses on the hierarchical control system for controlling the vehicles, and describes how planning and execution are done in the system.

2. Hierarchical Control

The control system for the vehicles is hierarchically structured and is shown in Figure

1 [Albus 87]. This control system is based on the one developed for the Automated Manufacturing Research Facility at NBS [Simpson et al. 83]. It is divided into three main components, shown as columns in Figure 4. These are *sensory processing*, *world modeling*, and *task decomposition*. The goal of the task decomposition component is to perform real-time decomposition of task goals by means of real-time planning, execution and task monitoring. The goal of the world modeling component is to accumulate and store information about the environment and the internal state of the system, and to make this information available to the task decomposition component. This component also simulates possible future states of the world. The database of information about the state of the world and the internal state of the system is kept in a common memory area.

The goal of the sensory processing component is to detect and recognize patterns, events and objects, and to filter and integrate sensory information over space and time. Information extracted from the sensory data is used to update the world model database.

The world model serves as a buffer between the sensory processing component and the planners and executors of the task decomposition component. That is, queries about the world required for planning and execution are made to the world model, and sensory processing is used to update this world model.

The control system is divided hierarchically into several levels. We view this kind of hierarchical division as a means of converging broad, high-level goals into commands to actuators, motors, communication transducers, sonar transducers, etc.

In the task decomposition hierarchy, the highest level, the *mission* level, converts a commanded mission into commands to each of a set of groups of vehicles. These commands

involve tasks that treat a whole group of vehicles as a single unit. The *group* level converts group commands into commands to each of the vehicles in the group. These commands involve large tasks for each vehicle. The *vehicle task* level converts task commands into elemental moves and actions for the vehicle. The *e-move* (elemental move) level converts elemental moves and actions into intermediate poses. These are converted into smooth trajectory positions, velocities, and accelerations by the *primitive* level. Finally, the *servo* level converts these into signals to actuators, transducers, etc.

3. Hierarchical Planning and Execution

Before describing the elements of hierarchical planning and execution, we will provide our working definition of a plan, and describe the difference between planning and execution. A plan is made up of actions and events. The events are either events in the world or events in the internal state of the system. We represent a plan as a graph (Figure 2). The nodes of the graph represent actions and the arcs represent events. The purpose of the planner is to obtain a plan graph. It can either generate it or retrieve it from a database.

We define execution as the process of carrying out a plan. The purpose of the executor is therefore to step through the plan graph. When the executor arrives at a node of the plan graph, it "executes" the action associated with the node. If an action is at the lowest level of the hierarchy, then executing it involves sending signals to hardware. Otherwise, executing an action involves sending it to a lower level where it can be decomposed. As the executor sits at a node of the plan graph, it monitors for events associated with arcs leading out of the node. This monitoring is done at a fast cycle rate. The process of monitoring for an event consists of querying the world model database for that event. If an event has occurred, the executor follows the arc corresponding to that event and steps to the next action.

The notion of hierarchical planning is shown in Figure 3. An action is first input to the top level as a task command. This task is decomposed both spatially and temporally. Spatial decomposition means dividing a task into logically distinct jobs for distinct subsystems. For example, the group level will have a

different planner for each vehicle in the group. Temporal decomposition means decomposing a task into a sequence of subtasks. The first step in the plan is then the input task to the next lower level, and this, in turn, is decomposed both spatially and temporally. At each successively lower level, the actions become more detailed and fine structured.

There are two primary reasons for the hierarchical approach -- to achieve real-time planning and control and to achieve understandability and programmability. At the higher levels of the hierarchy, actions are large scale and they take a long time to execute. Therefore, the search space used to generate plans is coarse and covers large space and time. At the lower levels, actions are smaller scale and they take a short time to execute. The search space is therefore fine and covers small space and time. As a result, the search spaces at all levels are small enough so that the search is manageable. Furthermore, all levels run in parallel.

The understandability and programmability comes about because the control system is decomposed into small modules whose functions can be well understood. Furthermore, different factors are taken into account at different levels, i.e., mission requirements, group tasks, vehicle tasks, elemental actions, etc. In this way, when new knowledge is added to the system, the modules in which this knowledge should reside are more apparent.

4. Levels in the MAUV Task Decomposition Hierarchy

4.1. Mission Level

The inputs to the mission level are a command and a mission value function. The command is a task involving a mission strategy, e.g., SEARCH-AND-DESTROY, SEARCH-AND-REPORT, and MAP. Associated with each command is a list of subtasks that define the command. The mission value function is a function used to score the mission, and is composed of the following elements:

1. *A value for each vehicle* -- used to assess the desirability of plan alternatives involving high risk to individual vehicles, or even the deliberate sacrifice of a vehicle.
2. *A value for each subtask* -- specifies the

importance of the successful completion of each of the subtasks.

3. *An information value for each subtask* -- specifies the importance of returning information collected while executing each subtask.
4. *A value of stealth for the mission* -- specifies the importance of avoiding detection by the enemy during the mission.
5. *The amount of battery energy available for the mission.*

The function of the mission level is to:

1. Subdivide the vehicles into groups. In our scenario, we have only one group, which contains two vehicles.
2. Determine whether any of the subtasks defining the input mission command should be omitted.
3. Provide a coarse description of routes and tactics for the mission that are sent to the lower levels.
4. Determine appropriate priorities to be used by the lower levels in planning the subtasks.

The outputs of the mission level are the group subtasks and priorities. Priorities are values indicating the importance of the following factors during lower level planning: time used, energy used, stealth, and vehicle survival.

The mission level planner attempts to generate an optimal sequence of subtasks as follows. First, a set of promising plan parameters is chosen. These include a specific sequence of subtasks and an estimate of the time and energy priorities. Next, the planner uses *outcome calculators* to determine the result of choosing these plan parameters. For example, the transit outcome calculator determines the projected risk and the time and energy consumption for each transit leg of the mission. In order to do this, the outcome calculator plans a coarse route. This route will eventually be passed to the lower level planners.

The results of the outcome calculators are then scored based on the mission value function which was input to the mission level. If the score indicates that a clearly satisfactory set of plan parameters has been chosen, then these are passed to the lower level. Otherwise, a new set of plan parameters is chosen and the procedure is repeated. If the time allocated to the planner

to make a decision has terminated, the best set of plan parameters thus far found will be passed to the lower level.

Replanning is done at regular intervals throughout the mission by repeating the program described above. If replanning results in a different plan from the one currently being executed, it is installed in place of the current plan. In this way, the world and vehicle situation is repeatedly evaluated so that the plan generated from the most recent information is always being executed. Further details about the mission level may be found in [Pugh and Krupp 87].

4.2. Group Level

The inputs to the group level are a command and a set of priorities. The command is a task involving multiple vehicles, e.g., TRANSIT, ATTACK, RASTER-SEARCH. The priorities are values indicating the importance of stealth, destruction, time, and energy. These priorities will be used as weights in the cost function during A^* search [Nilsson 71].

The input group tasks define large scale actions to be performed by groups of MAUV vehicles. The function of the group level is to decompose these into sequences of tasks for individual vehicles. This level also attempts to maximize the effectiveness of the whole group by scheduling the actions of each vehicle so as to coordinate with the other vehicles in the group.

The planner uses A^* search during planning. The following factors are used in the cost function for this search:

1. *Probability of traversal.* This is based on known obstacles (such as large land masses) and known density of clutter (e.g., a group of small islands in a given path would result in a low probability of traversal).
2. *Probability of detection* by enemy sonobuoy fields or by enemy ships containing acoustic sensors.
3. *Probability of destruction* by enemy minefields or enemy ships containing active sonar sensors.
4. *Energy used.*
5. *Time used.*

6. *Deviation penalty from path specified at level above.* The input task command to the group level may specify a path to be followed. This path is taken into account by the cost function by means of a deviation penalty.

The outputs of the group level are the vehicle tasks and priorities. The output priority values are the same as the input priorities.

4.3. Vehicle Level

The inputs to the vehicle level are a command and a set of priorities. The command is a task performed by a single vehicle, e.g., GOPATH, WAIT, RASTER-SEARCH, LOCALIZE-TARGET, RENDEZVOUS. The priorities are the same as the input priorities to the group level.

The function of the vehicle level is to decompose the input vehicle task into a sequence of tasks for each subsystem of the vehicle. These subsystem tasks are called elemental moves or actions (e-moves). We consider three subsystems, the pilot, sensors and communications subsystems.

The pilot planner uses the world model database to search for a path between the start and goal positions indicated by the input vehicle command. A* search is used and its cost function has the same factors as used at the group level.

The communications planner schedules the messages to be sent. Currently, this schedule is extracted from a rule database. In the future, the schedule will be determined by computing the value of each message, its urgency, the risk of breaking communications silence, and the power needed to transmit the message.

The sensors planner schedules the activation and deactivation of passive and active sonars. Currently, this schedule is also extracted from a rule database. In the future, the schedule will be determined by computing the value of taking sonar soundings, its urgency, the risk of breaking silence for active sonar, and the power needed to take the sonar soundings.

The outputs of the vehicle level are the e-move tasks.

4.4. E-move Level

The input to the e-move level is a command which is an elemental move or action involving a single subsystem, e.g., GO-STRAIGHT (pilot subsystem), ACTIVATE-ACTIVE-SENSOR (sensor subsystem), SEND-MESSAGE (communications subsystem).

The function of the e-move level is to decompose the input e-move command into a sequence of low-level commands to the particular subsystem controller.

The pilot e-move can be defined as a smooth motion of the vehicle designed to achieve some position, orientation, or "key-frame pose" in space or time. The pilot planner at this level computes clearance with obstacles sensed by on-board sonar sensors and generates sequences of intermediate poses that define pathways between key-frame poses. A* search is used to generate these paths. The cost function used during this search uses the following factors:

1. *Traversability.* This is based on known local obstacles. The traversability of a given path is either 1 (the path is traversable) or 0 (the path is not traversable).
2. *Distance travelled.* A shorter path is always preferred. This helps obtain smooth final paths.
3. *Deviation penalty from path specified at level above.* As in previous levels, the input command to the e-move level may specify a path to be followed. This path is taken into account by the cost function by means of a deviation penalty.

A communications e-move is a message. The communications planner at this level encodes messages into strings of symbols, adds redundancy for error detection and correction, and formats the symbols for transmission.

The sensors e-move is a command to activate or deactivate a passive or active sonar. The sensors planner at this level decomposes sonar activation commands into a temporal pattern of sonar pings.

The e-move level is the lowest level we currently consider in the MAUV architecture. The outputs of this level are low-level commands to the subsystem controllers of the MAUV vehicles.

5. Cooperative Vehicle Behavior

Cooperative behavior between the two vehicles is achieved as follows. The vehicles start out with identical software, except for the vehicle identifier, which is unique for each vehicle. This implies that each vehicle has a mission and a group level, and mission and group level planning is done on both vehicles. If the two vehicles sense the exact same world all the time (i.e., they receive the same sensor input), then mission and group planning will be identical between the two vehicles, and they will achieve coordinated behavior. This is because the two vehicles will generate identical plans for both vehicle 1 and vehicle 2, and each vehicle will simply execute the appropriate plan for itself.

If, instead of always having identical world model databases, the vehicles have the same world model information with regard to significant world properties (i.e., properties relevant to generating and executing mission and group level plans), then mission and group planning will still be identical between the two vehicles. This is the method we currently use to achieve cooperative behavior. The significant world properties relevant to our scenarios are the positions of large land masses such as islands, the positions of sonobouy and mine fields, the positions of the two vehicles, and the positions of enemy targets and defenses. Islands, sonobouy fields and mine fields are input at the beginning of the mission and do not change. Therefore information about these will be identical in the vehicles' world model databases. In order to ensure that information about the other significant world properties are the same in both databases, each vehicle, upon detecting a new target or defense, immediately communicates this to the other vehicle. In addition, each vehicle regularly communicates its position to the other vehicle.

A problem with this technique of achieving cooperative behavior is that, as the scenarios become more complex, more information would have to be regularly communicated between the vehicles. In addition, if a group had many vehicles in it, regular communication from each vehicle to all the others would have to occur. An alternative technique which seems more promising is to designate one vehicle in each group as group leader, and to designate one vehicle as mission leader. The

mission leader performs mission planning and communicates the plans to each group leader. Each group leader does group planning and communicates the plans to the individual vehicles in the group. In this way, if different vehicles have different world model databases, they will nevertheless execute cooperative maneuvers determined from the world model databases of the group and mission leaders. If communication cannot occur because of stealth requirements or because a vehicle is out of communication range, then each vehicle still has mission and group level software and can generate its own plans. Of course, this could lead to non-cooperative maneuvers. Once communication is re-established, the mission and group leaders can take over.

6. Real-Time Planning

This section describes the real-time planning system used at the group and vehicle levels of the hierarchy. The block diagram in Figure 4, which shows this planning system, can be applied to the group level as well as the vehicle level. An input task command first goes to the Planner Manager, which contains two modules. The first, the Job Assignment Module, divides the input task into several jobs and sends each to a different planner. The different planners then work on these jobs in parallel. The second module, The Plan Coordination Module, coordinates planning among the various planners. Currently, this coordination is accomplished by generating constraints to be met by all the planners. For example, if each planner corresponds to a separate vehicle, this module might generate constraints consisting of a position where all the vehicles are to rendezvous and a time when this is to occur. Each individual planner would attempt to meet the constraints. If one of them could not, it would report back to the Plan Coordination Module which would then generate a new set of constraints. In the future, the Plan Coordination Module will also coordinate communication among the planners. Some constraints can be determined only by the planners at plan time, and these would have to be communicated to the other planners. For example, one vehicle planner might want as part of its plan one of two actions depending on what another vehicle planner generates.

After a planner has finished generating a

plan in the form of a plan graph, the executor associated with the planner steps through the graph.

Each planner contains several modules (Figure 4). The Cyclic Replanning Module accepts an input command (or job) from the Planner Manager and, at regular cycle times, generates a new plan. The primary way in which our system performs replanning is by generating new plans regularly. The standard way of doing replanning is to post some simple conditions on the world which, when met, causes replanning to occur. Our approach, however, is based on the notion that the best way to know whether the world has changed in such a way as to require a new plan is to actually run the algorithm that generates the plan, and then to see whether the plan has changed. The advantage of doing it this way rather than posting some simple conditions is that there could be a complex interaction of events in the world that would require a new plan, and this complex interaction is exactly what the planning algorithm looks for and evaluates.

One issue that must be considered is real-time planning and how it is handled by the planner. As stated above, we view a plan as being composed of actions and world events. Execution of the plan by the executor occurs by monitoring for world events and stepping to the appropriate action based on which world events have occurred. Let t_1 be an arbitrary point in time and let E be the set of events in the world occurring at t_1 . We define *real-time planning* as the process of generating plans quickly enough so that there is always an action a given to the executor such that

1. action a is part of a plan p , and
2. plan p represents an "appropriate" response by the system to events E at time t_1 .

Let t_1 be as defined above and let t_2 be the latest time by which an action must be executed in order to appropriately respond to the world events E . Then the *planning reaction time* is defined as the time interval $t_2 - t_1$.

Fortunately, the planning reaction time is different at different levels of the hierarchy. At the higher levels, the world representation is coarse, planned actions occur over large time scales, and world events are coarsely represented. Therefore the planning reaction time of the system can be relatively slow. At

the lower levels, the world representation is detailed, planned actions occur over small time scales, and world events are represented in detail. Therefore the planning reaction time must be fast.

The cyclic replanning time at each level is determined by the planning reaction time. The cyclic replanning times at the higher levels are longer than at the lower levels. At the end of a cyclic replanning time interval, the next action to be taken must be determined by the planner, for the executor must always have an action to carry out. However, these time intervals will often not be enough for the planners to generate new full plans. Therefore, the planner will pass on to the executor whatever is its best plan at the end of the cycle time, even though the planner may not have finished planning to completion. In our implementation, where A^* search is used, the best plan at any point in time is the path in the search tree from the root to the leaf node with lowest cost.

When the Cyclic Replanning Module has generated a new plan, the plan is passed to the Plan Update Module (Figure 4), which updates the Plan Graph.

If a subtask (i.e., an action) of the current plan is sent by the executor to the level below and the subtask cannot be achieved, then a signal is returned to the current level and the plan is modified by the Subtask Failure Replanning Module (Figure 4). Associated with each subtask command sent to the level below is a set of failure constraints. If these constraints cannot be met, then the subtask fails. Examples of failure constraints are (1) achieving the subtask within a time window, (2) achieving a goal (e.g., arriving at a given point in space), and (3) not deviating more than a certain amount from a given path.

The Subtask Failure Replanning Module has thus far been implemented only at the e-move level to handle imminent collision between the vehicle and the lake bottom. The module generates a plan in which the vehicle slowly moves upward, collecting sensory information, until it has determined that there is room to continue forward.

Both the Cyclic Replanning and the Subtask Failure Replanning Modules tap into the Plan Schema Database to generate plans. Plan schemas are used to define the input task commands and will be described next.

7. Plan Schemas

A plan schema is used to define a subtask command. It provides all possible sequences of actions that define the command. In order to determine the best sequence in a given situation, it allows the application of a cost function and provides the ability to perform a search which is driven by the plan schema. As shown in Figure 5, the plan schema is represented as a graph. The nodes of the graph represent actions and the arcs represent events in the world or internal events in the system. The plan schema is converted into a specific plan by an interpreter which steps through the plan schema graph and outputs a plan graph. When the interpreter reaches a node in the plan schema graph, it adds the action associated with the node to the output plan. It then queries the world model about the world events associated with the arcs leading out of the node. The queries relate to a hypothetical future world formed by starting with the current model of the world and simulating all the hypothetical actions in the output plan. The interpreter follows the arc whose world event is true, and then processes the next node in the plan schema.

The node of the plan schema is divided into two components, the *alternative action* component and the *context subroutine* component. The alternative action component contains a function that generates all possible alternative actions that can be considered when the node is reached. These alternative actions represent the possible operators that can be applied to the state space at a given point in the state space search. In Figure 5, for example, the GO-STRAIGHT node contains a function that returns all permissible directions for a GO-STRAIGHT action. Since the state space in this case is a three-dimensional grid, all GO-STRAIGHT actions, when executed, will lead to some adjacent point on the grid.

The context subroutine component of a plan schema node contains a subroutine that sets the context (i.e., sets certain variables) for the alternative action component. This context is also applied to all future nodes of the plan schema that will be traversed by the interpreter, even though these nodes also have their own context subroutine components.

The plan schema contains two types of arcs. The first type is a *world event* arc. This

arc contains a predicate that queries the world model about a hypothetical future world. A function is then applied to the result of this query, and the predicate returns true or false depending on the value of the function. In Figure 5, for example, the arc out of the GO-STRAIGHT node labeled "@POINT P" is a predicate that queries a hypothetical future world, resulting from the hypothetical execution of a set of GO-STRAIGHTs, about whether the vehicle is at point P. If it is, then the interpreter will step to the HOVER node.

The kind of predicate just described is a *plan time* predicate. Also associated with each world event arc is an *execution time* predicate. This is the predicate that is actually placed in the plan graph, and this predicate will query the most current world model at execution time.

The second type of arc in the plan schema is the *else* arc. This arc also contains plan time and execution time predicates. The plan time predicate returns true if the node that it leads out of has been processed and the predicates of all other arcs leading out of the node return false. In Figure 5, for example, there is an else arc and a world event arc leading out of the GO-STRAIGHT node. If the node has been processed and the predicate of the world event arc (i.e., whether the vehicle is at point P) returns false, then the predicate of the else arc will return true and the node will be revisited. The execution time predicate of the else arc returns true if the node that it leads out of in the plan graph has successfully completed execution and the predicates of other arcs leading out of the node return false.

References

1. Albus, J. S. "A Control System Architecture for Intelligent Machine Systems." *IEEE Conf. on Systems, Man, and Cybernetics*, Arlington, VA, October 1987.
2. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.
3. Pugh, G.E. and Krupp, J.C. "A Value-Driven Control System for the Coordination of Autonomous Cooperating Underwater Vehicles." *Proc. Fourteenth Annual Symposium of the Association for*

Unmanned Vehicle Systems, Washington, D.C., July 1987.

J.S. "The Automated Manufacturing Research Facility of the National Bureau of Standards." *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1983.

4. Simpson, J.A., Hocken, R.J., and Albus,

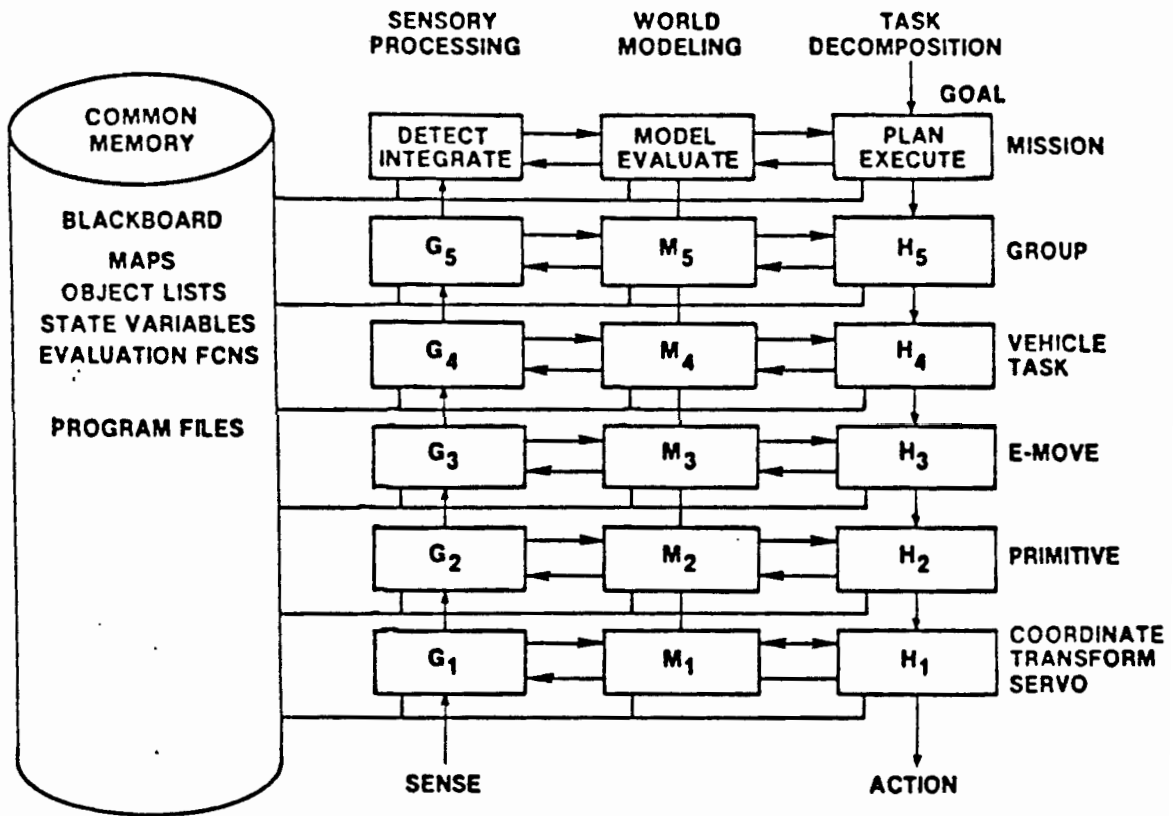


Figure 1: A block diagram of the NBS MAUV Control System Architecture.

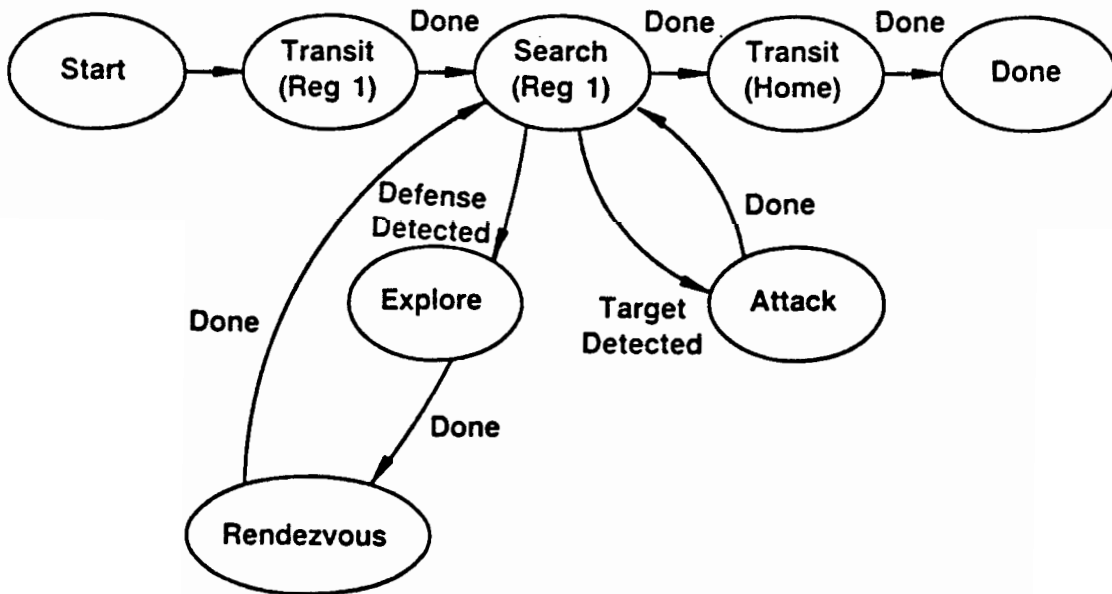


Figure 2: A plan graph for a mission level plan.

Hierarchical Planning

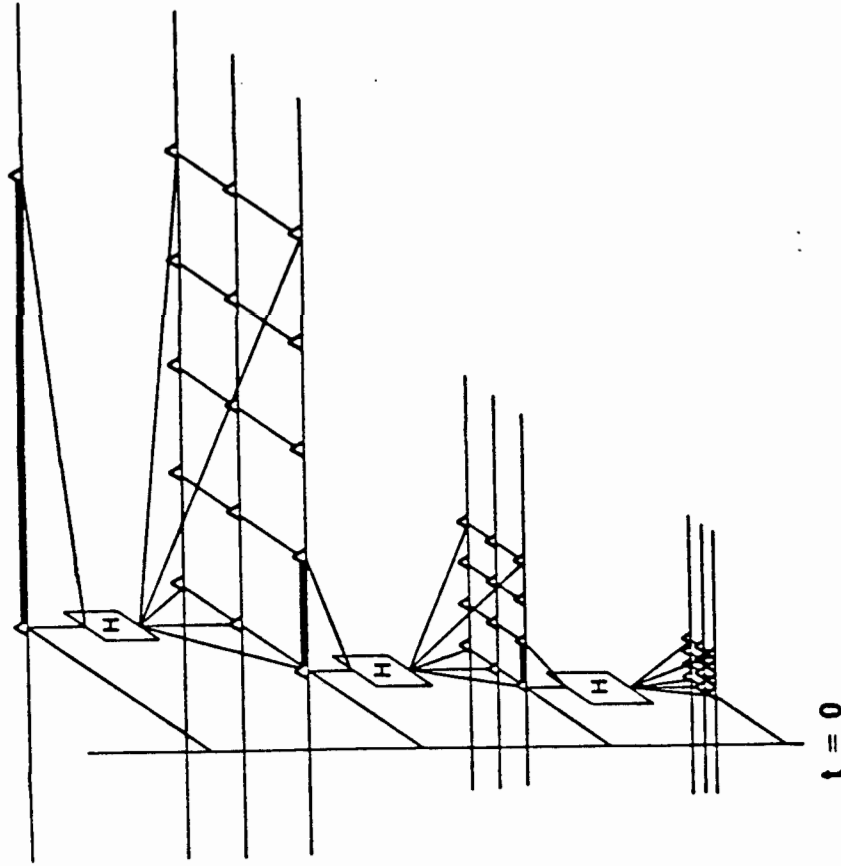


Figure 3: Three levels of real-time planning activity.

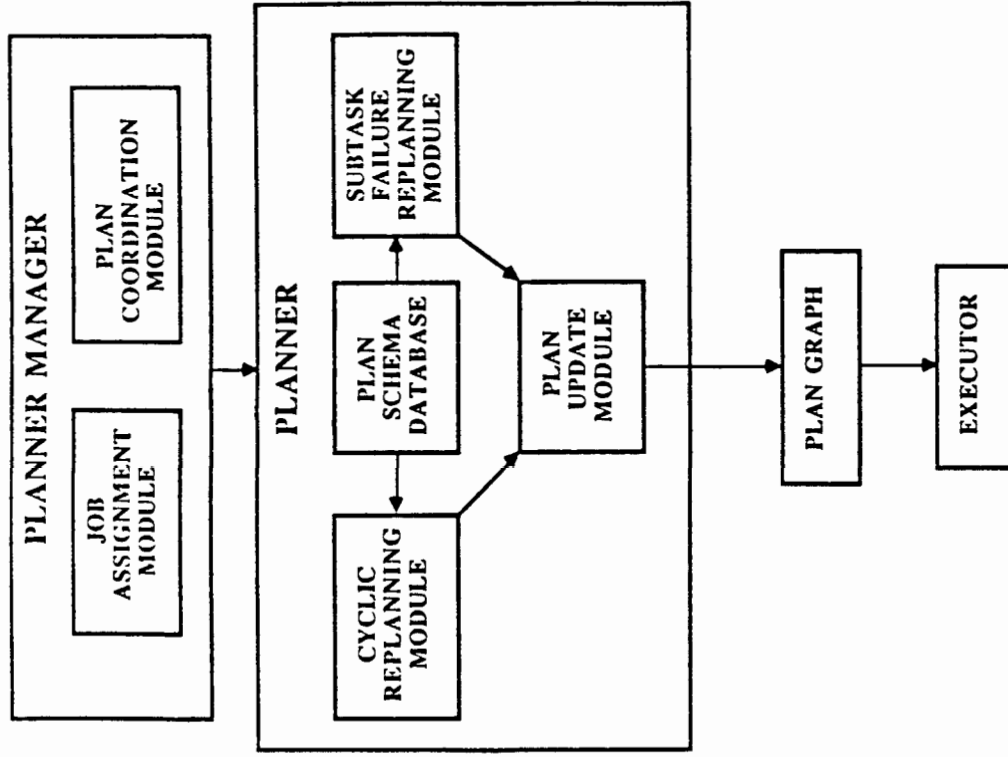


Figure 4: Group and mission level planner.

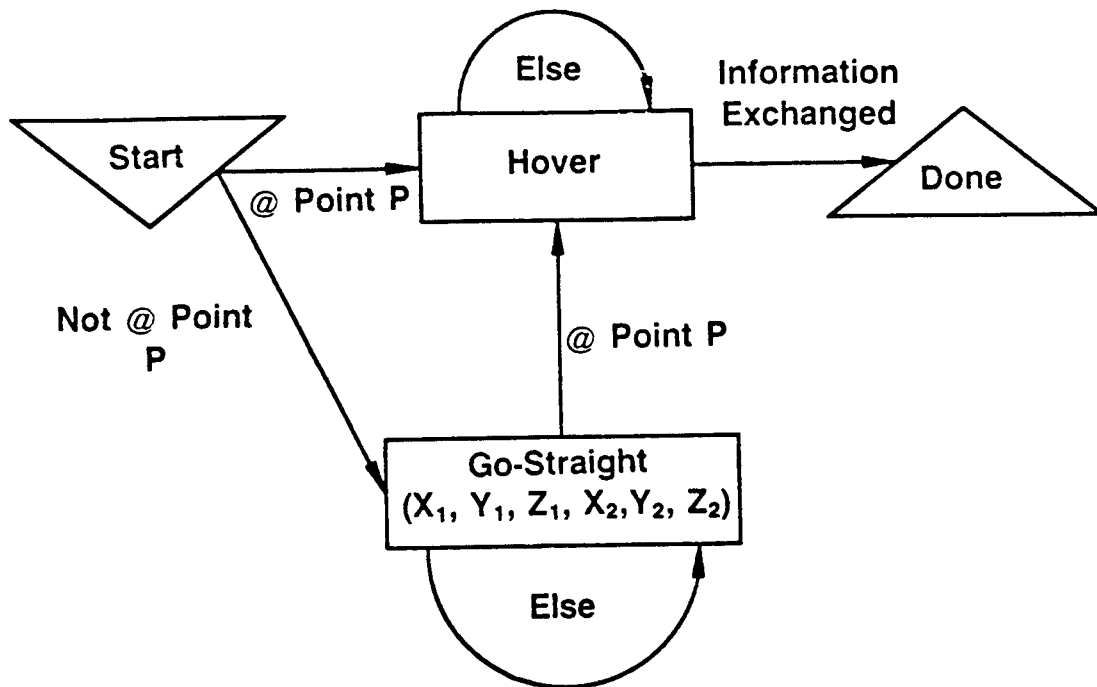


Figure 5: Vehicle level plan schema for "Rendezvous at Point P."