Planning techniques survey : their applicability to the mobile servicing system

Government of Canada | Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT **DOC-CR-SP-88-005**

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: Planning Techniques Survey: Their Applicability to the Mobile Servicing System

AUTHOR(S): D.L. Deugo, F. Oppacher, D. Thomas

ISSUED BY CONTRACTOR AS REPORT NO:

PREPARED BY: Carleton University
School of Computer Science

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 31098-7-21600

DOC SCIENTIFIC AUTHORITY: Peter Adamovits

CLASSIFICATION: Releasable

DATE: April 1988

2. Planning Techniques Survey:

Their Applicability to The Mobile Servicing System

April 28, 1988

Prepared By

Dwight Deugo
Franz Oppacher
Dave Thomas

School of Computer Science
Carleton University
Ottawa Ontario, Canada

Contract Serial No. 31098-7-2160/01-SW
Space Station Project Office
National Research Council

Scientific Authority P.J. Adamovits
Communications Research Centre

Executive Summary

A problem faced by the Mobile Servicing System (MSS) is the efficient scheduling of various operational tasks and the management of resources to perform those tasks.

There are many different Artificial Intelligence (AI) approaches to scheduling(planning). In this report, the authors present the generic planning paradigm and provide a survey of the different AI approaches to planning. These approaches include: Resolution planning, Hierarchical planning, Least-Commitment planning, Constraint Posting planning, Goal Directed planning, and Case-Based planning. For each approach, the authors describe the method, provide an example from the AI literature, discuss its advantages and disadvantages, and describe its suitability for scheduling on the MSS. A survey of NASA-related scheduling systems and other expert systems designed for scheduling is also included.

The report concludes with a proposed approach for scheduling tasks and the management of resources on the MSS. By using the suitable AI planning approaches and Dynamic Memory Theory, a robust, dynamic, and efficient planner is created. This planner handles resource constraints, feedback, and operator input, enabling it to handle the dynamic environment of the MSS.

TABLE OF CONTENTS

1.0 INTRODUCTION

This paper provides a survey of the current approaches to planning and discussions of their appropriateness for the Mobile Servicing System (MSS). The MSS is a planning system performing resource management for Space Station. The purpose of the survey and discussions are to aid in the investigation of the development of an intelligent planning assistant which can handle the dynamics needed to cope with the MSS.

By doing a review of these approaches, our approach can extend the related work on planning in Artificial Intelligence(AI) in various respects. In particular, our approach will attempt to handle resource constraints and feedback and achieve some robustness through plan learning using dynamic memory techniques. A major goal of any planning system is to achieve some degree of autonomy. For example, since robots are not fully accurate and may have to operate in a changing environment, an autonomous mobile robot must rely on feedback. The presence of feedback presupposes an ability of dynamic replanning. Feedback tasks often involve tight time and other resource constraints and require the need for dynamic memory techniques to meet these constraints.

We begin with the general MSS problem description to give the reader a feel for the planning problem. Next, we introduce the generic planning paradigm by discussing the planning process, the inputs and outputs to the planning process, and the related problems of planning in what is known as "The

Block World". We continue with a survey of the following planning techniques:

1. Resolution Planning
2. Hierarchical Planning
3. Least-Commitment Planning
4. Constraint Posting Planning
5. Goal Directed / Immediate Execution Planning
6. Case-Based Planning

Each section includes a discussion on the following: the planning technique's method, examples, its advantages, its disadvantages, and its suitability to the MSS.

Next, we discuss current scheduling systems. This survey includes a brief description of each scheduling system and a list of preliminary design concepts derived from them.

We conclude with a recommended approach for the development of an intelligent planning assistant. Our approach requires, in addition to some of the planning techniques described, dynamic memory techniques to support the dynamics need to cope with the MSS.

2.0 THE PROBLEM

In this section, the MSS problem is outlined. This section is provided to familiarize the reader with the MSS problem. In a later section we will propose a planning approach to solve the MSS problem after reviewing various planning techniques and their suitability for solving it.

The MSS problem consists of the following components:

- A Task - This, as the name suggests, is a description of a unit of work that has been identified and parceled for the MSS to execute. Each task, as well as identifying a list of actions and operations to be performed by it, also includes a list of demands for resources such as heat and time.

- Resources - These are physical supplies of resources such as time, heat, and power, available to the MSS. The supply of these resources can be constant or variable and usually depends on time.

- Constraints - These are operator or planner imposed limitations on the supply and demand of resources.

- Operator - This person interacts with the MSS to provide it with the resource, constraint, task, and planning information it requires or makes modification on such data as needed.

- Plan - An ordered list of tasks to execute.

- Task Executor - This is a process that attempts to execute a task or collection of tasks using a plan.

- Sensors - These are sensors that monitor and feedback resource supply information while the Task Executor is processing the tasks.

Figure 1 shows a block diagram of the components connected together.

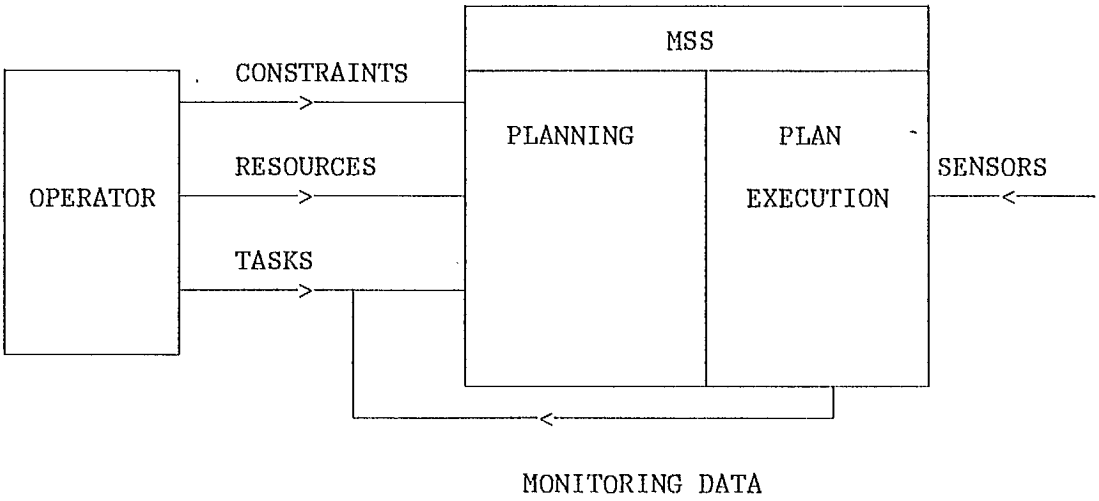

Figure 1. MSS Components

We now define the MSS problem using these components. The operator, or information entry and control component(IECC), receives a request to plan how and when it can schedule and execute 'n' tasks on the MSS. To generate a plan, the IECC must identify the demand for resources each task requires

and the availability of those resources. From this information and by applying its own constraints, the IECC develops a plan that will allow the tasks to execute. While the tasks are executing, the IECC also monitors the MSS to insure the plan is executing successfully; it may not due to an error in the plan or because of unexpected changes in the supply of resources indicated by the sensors. If this occurs, the IECC can perform corrective maintenance by altering the plan, stopping the execution of various tasks in the plan, or stopping the execution of the plan and starting over.

The environments of the planning activity and the execution of the plan are very dynamic. Each environment has many variables that are likely to change through the course of each process. There are initial constraints and those that are imposed by the MSS as the plan is executed. The IECC must be in full control to insure that all tasks are executed successfully. The IECC constantly uses planning information that has been previously stored in past planning sessions and tries to remember the variable information to produce the correct plan for the current situation.

In addition to the environment being dynamic, so too must be the IECC and its plans. If a plan is about to fail, the IECC must take some action. However, the complexity of the domain makes this maintenance task a formidable one.

Planning is an easy task in a small domain. However, when the number of variables in a plan are increased, the number of resources is large and can vary, especially with time, and the list of a task's demands for resources

is large, then the IECC can have a difficult time in producing correct, efficient plans.

The problem simply put is: how can we automate part of the planning process to help the IECC in its task and still leave it in the control loop playing an important role in overseeing the planning task? After surveying the current approaches to planning, we will propose an approach to answer this question.

3.0 GENERIC PLANNING PARADIGM AND THE BLOCK WORLD

The ability to plan ahead is an essential aspect of intelligent behavior. By being reminded of past situations, the actions taken in those situations, and the consequences of those actions, we can achieve our goals by exploiting that information to formulate new plans to anticipate past dangers or problems, or we can use old plans to achieve similar goals, thereby economizing on our resources.

When planning, we begin with an initial state, an initial set of properties, and try to produce a plan that results in a goal state and new sets of desired properties. The input data to the planner include: an initial state, a desired goal state, a set of possible actions, and a database of information about the initial state and all other possible states. The output of the planner is a set of actions that, when executed in the current state of the planner, produce the desired goal state and description.

We now discuss the inputs and outputs of the planner in more detail.

3.1 STATE

A state is an environment in which actions are executed. Each state definition includes: a descriptive definition of the state, a list of possible actions available for that state, and a list of common actions available to all states.

The state in which the planner is expected to begin operation is called the initial state. The state where the planner is expected to end operation is called to goal state. All other states are known as intermediate states. Execution continues throughout the intermediate states leading to the goal state.

The sample "Block World" considered here consists of two blocks A and B, a table, and a robot arm. State descriptions include various block positions such as ON(A,B) - block A is on block B, HOLDING(A) - the robot is holding block A , and ONTABLE(B) - block B is on the table.

3.2 GOALS

A goal is a definition of the desired end state. It is not necessarily a single state, but rather it is a precise definition of what conditions lead to the termination of the planning process. For example, in the block

world the goal state may consist of having two blocks stacked one on top of another. If there are two different blocks, this leads to two possible stacking arrangements as well an infinite number of table positioning possibilities.

This leads to the conceptualization that goals are unary relations on states. We say that a state is a goal state if and only if it satisfies these unary relations. For example, the relations ON(A,B) AND ONTABLE(B) states we have reached the goal state and can stop processing when block A is on block B and block B is on the table.

3.3 ACTIONS

Actions consist of primitive or composite actions that can be used to convert the initial state to some other intermediate or goal state. There can only be a finite number of primitive actions, but there can be an infinite number of composite actions in the set. Actions are also known as events, and they are discrete transformations of states in the world of the planner.

By using defined sets of actions, we can restrict the planner to produce types of plans that can be executed by some form of executor. For example, it would be silly to provide a conditional action to check the size of an object if the executor does not have the capability.

An action description includes: operator descriptions and frame axioms for the primitive actions, definitions of composite actions, and constraints that must be true of the state in which the action is to be performed. Frame axioms are descriptions of the what remains the same during the action. For example, the following describes the stacking operation of two blocks which is defined as the stacking of block X on block Y.

```
Stack(x,y) :=
        Precondition --> CLEAR(Y) AND HOLDING(X)
        Action     --> ON(X,Y)
```

To stack block X on block Y, the precondition states that the block Y must not have any block on top of it and the robot arm must be holding block X; the action states to the robot must put block X on block Y. We can only execute the action if the preconditions are true.

3.4 PLAN

A plan is a set of proposed actions to be taken from the initial state. After executing the actions in the initial state and subsequent intermediate states, the executor will finish in the goal state satisfying the goal conditions.

3.5 PROBLEM AREAS

Now that the planning process and its terms have been identified, we switch our attention to areas of planning that create problems for the planning process. These problems include: planning direction, unachievability pruning, state alignment, the frame problem, and real-world problems.

3.6 PLANNING DIRECTION

One of most important determinants in the efficiency of the planning process is the planning direction. The problem, simply put, is whether it is better to plan forward from an initial state or to plan backwards from a goal state. On some occasions it may be better to use a combination of the two. To determine the best approach, a planner should examine the number of possibilities to be explored in both the forward and backward directions, and it should calculate the branching factor in each direction.

Once the general direction has been established, a planning process must still attempt to get the plan heading in the direction of the solution. This will minimize the wasted effort used in planning away from the solution. One method of achieving this is through goal directed processing. The method called "Prove-As-You-Go" (Winslett 1987) adds preconditions to all of its operators. These preconditions are used to insure that the per-

formed operations are not in violation of the goal of the action. Another method called "prove-ahead" (Winslett 1987), finds all the possible action effects ahead of time and acts to prevent the undesirable ones.

By effectively guiding the planning direction, efficient plans and a reduction in the plan validation phase can both be achieved.

3.7 UNACHIEVABILITY PRUNING

One source of computational waste in backward planning is the work done in states that are unachievable. For example, the state where block A is on block B and block B is on block A. When this state is reached, the planning process should be immediately recognized it as being invalid and prune it from any further consideration.

3.8 STATE ALIGNMENT

In some planning methods, situations are encountered where several conditions must be satisfied in a single state. When operator description axioms are used to satisfy one of these conditions, we end up with a subproblem in

which the operator's preconditions must be met in one state whereas the remaining conditions must be met in the successor state.

State alignment attempts to avoid reductions of conditions in one state while there are still conditions on a successor state to be reduced. By doing this, inconsistent conditions can be identified and eliminated from further consideration. Planning efficiency can also be improved when conditions can be collected into a single state and executed in parallel.

3.9 THE FRAME PROBLEM

As operations are performed and transitions are made from one state to another some facts(state descriptions) become true and others false. However, nothing is stated about facts that were true beforehand and still are, or about facts that were false before and still are.

The problem of characterizing the components of a state that are not changed by an action is called the frame problem(Hayes 81). The problem is how to differentiate those items that remain unchanged by an action from those that are? One way of doiug this is to write frame axioms that describe the properties that remain unchanged after the execution of an action.

The number of required frame axioms is typically proportional to the product of the number of relations and operators in the planner. In real-world problems, the number of frame axioms can grow quite large and complex. Further-more, only a few are used in any one state at any time; this means one must describe what is usually not happening in addition to what is.

3.10 REAL-WORLD PROBLEMS

The bulk of AI research on planning has not centered around what we like to call real-world problems. This is a problem in itself. The real world has constraints such as cost, speed, and efficiency. Planners that take hours to run, require large quantities of memory, cannot cope with fluctuating resources supplies, time-varying variables, and inaccuracies of physical components, and produce large plans are not usually desired in this world.

Another problem arises from the fact that most planners start without an initial plan. The planner must always attempt to build a new plan from scratch using the base environment as a starting point. Real-world plans involve using partially existing plans in cooperation with a new plan to achieve a goal state. It is a rare situation when an actual planning activity starts from square one. Real-World plan must be: Plans must also be dynamic, allowed to change, and not have to start the planning process always from scratch.

4.0 PLANNING SURVEY

4.1 RESOLUTION PLANNING

4.1.1 METHOD

Green's method of planning is based on resolution. This method takes as inputs: an initial state, a unary relation constant which is called the goal relation, a predicate satisfied by the plan's execution, and a database of facts about the initial state. These facts include the goal relation and the available operators.

By using fill-in-the-blank resolution to derive a plan, the side effect is the existence of a correct plan. The passed predicate is used to check every answer returned by the process. If the answer satisfies the predicate, it is returned as an overall answer to the plan.

Fill-in-the-blank resolution is the binding of free variables in a predicate-calculus sentence to a database sentence that logically implies the original sentence. This is achieved by substituting the bindings into the original sentence.

The key component to this type of planning is the use of symbolic logic, both as a tool for programming and for characterizing structures of actions or events. Logic operators are used to make the transformation from one world-state to another. The primitive operators or complex relations may be composed of operators such as "and", "or", "not", "if- then", or "if and only if".

4.1.2 EXAMPLES

The classic example of this type of planner is STRIPS (Nilson 1980). STRIPS represents states of the world as conjunctions of first order logic facts. Given an initial state and a goal state, STRIPS searches through a space of states to generate an ordered set of operators that if executed would achieve the goal states.

STRIPS attempts to solve the frame problem by adding the notion of a pre-condition list, an add list, and a delete list. Each item in the list is expressed as a first-order formula. This approach provides a means of describing how the world would change as a result of performing an action. This enables the state of the world to be adequately described after the action has been performed. It also insures that the correct state of the world is obtained before an action is performed.

(Sandewall & Ronnquist 1986) use a form of temporal logic for their system's action structures. This is based on a temporal ordering of points in time. This language allows the planner to say things like "in the resulting state after first doing A, then doing B and C in parallel, the proposition P will hold".

(Sandewall & Ronnquist 1986) also introduce the concept of prevail-conditions. These are conditions that must hold for the full duration of the action. This is a key development in allowing actions to proceed in parallel. By defining what conditions must hold for the duration of the action structure, actions not affecting those conditions can proceed while others must wait for the current action to complete.

(Georgeff, Lansky and Bessiere 1985), (McDermott 1985), and (Allen 1981) all consider an action or event to be a set of sequences of states and describe a temporal logic for reasoning about these actions or events. (Georgeff, Lansky and Bessiere 1985) in addition, consider various "mental entities" such as beliefs, goals, and intentions in its sequences of world states. These entities, called processes, can be executed to generate a sequence of world states called the behavior of the process.

4.1.3 ADVANTAGES

The only real advantage to Resolution planning is that it will guarantee to produce a correct plan if one exists. This is due to the resolution based method which makes it possible to prove some strong properties about the planner's abilities.

The examples provide insights into other characteristics planners should have. The ability to do parallel actions greatly enhances the efficiency of a plan's execution. Any type of planner should attempt to provide this capability. Including goals, beliefs, intentions, and physical characteristics, as part of the state description, enhances the world description. These enhanced descriptions allow the planner to take more real-world situations into consideration. It is goals, beliefs, intentions such as "I want the block here because is fits better", and physical descriptions such as where the block is located, that compose real-world problems. Because enhanced descriptions are certainly part of real-world planning problems, using them to infer and index past solutions and situations will allow real-world applications to be considered.

4.1.4 DISADVANTAGES

Resolution planning is generally inefficient. The backtracking involved in finding the existence of a correct plan can be an enormous task in large systems. Some form of focusing or pruning mechanisms of invalid states is needed to improve the efficiency.

Resolution planning relies on very static situations. Adding new axioms, actions, or world facts as the plan proceeds can cause the plan to become invalid. It does not handle a dynamic environment with which most real-world plans must deal.

The attempt by STRIPS to solve the frame problem creates an increase in the amount of initial information that must be provided for the state transitions. STRIPS requires information for action preconditions, deletion lists, and add lists. Therefore, STRIPS spends most of its time gathering information used for trying to identify these list items, that for the most part are not used. STRIPS may be able to work in the block world, but for real-world applications the knowledge representation becomes to large and complex. This leaves the application programmer the frustrating task of attempting to predict the state of the world before and after every operation in every possible situation.

Plans by their very nature depend on the meaning of an expression in every situation. Unfortunately, in ordinary logic the meaning of an expression does not change from situation to situation. Situational logic, where one

can refer explicitly to situations or states of the world, tries to fix this difficulty. However, it does not solve the problem in all situations. Consider a monkey that is presented with two boxes, and is informed that one contains a banana and the other a bomb but is not told which. His goal is to get the banana, but if he goes near the bomb it will explode. The problem should have no solution. However, in situational logic a proof can be found. The monkey would consider if it were to go to the box with the bomb would it get a banana, and if it could not it would then go to the next box. This plan is not executable because it allows the monkey to consider whether a given fact, is banana here at the box with the bomb, is true. This of course is not possible because the monkey would explode while attempting the maneuver.

(Manna and Waldinger 1987) attempt to overcome this limitation. They use two classes of expressions. The static or situational expressions denote particular objects, states, and truth-values. For example, Put(state-1, loc-block-a, new-loc-block-a), is a static expression describing the block Put operation in state-1. Fluent terms, which do not denote any particular object, truth-value, or state, designate such elements without respect to a given state. For example, Put(a, a'), is an expression designating a block, truth-value, or state without respect to a given state (where a and a' are themselves fluent terms that designate blocks). States are linked to fluent terms through the use of a link operator, for example, s;Put(a, a'). This permits the execution of an operator for an explicit state "s".

By not allowing fluent terms do refer to states explicitly, the knowledge of the agent will be restricted to the implicit current state. In this way,

it can ensure the plans extracted may be executable. This is because it will be unable to tell, for example, where the location of a given block is in a future state.

4.1.5 MSS SUITABILITY

Resolution planning is not suitable for our particular problem because of many reasons: for each planning task, no matter how similar it was to a successful past plan, it must plan from scratch; it must have all knowledge of the world explicitly declared; the plan cannot be adapted; and there is no outside manipulation of the planning process from another source once the planning process has started.

Resolution planning expects exact initial and goal state definitions, and it does not allow the world definitions to vary as the planning process proceeds. The MSS has to function in a very dynamic world;, new constraints are created constantly, and old ones are removed. This would cause a Resolution the planner to restart the planning process every time the world changed and is not an acceptable approach.

Resolution planning is also very knowledge intensive. Actions must specify: what preconditions are present before an action can take place, what new conditions are to be inserted into the world definition after the action is performed, and those conditions to be removed when the action is completed.

In all but small domains, such as the block world, this type of information becomes quite complex and unmanageable.

Once the knowledge has been installed into the planner, under the same conditions the planner will always produce the same plan. If the plan is bad, one would like to have it changed, but this planner does not have the facilities to adapt itself. The MSS must be able to adapt its plans especially since its environment is constantly changing.

The MSS uses an operator to interact with the planning process. However, Resolution planning does not allow for any intervention by the operator or feedback from the sensors once the planning process has begun. This information can be of great value for the planner especially when it gets stuck. By permitting interaction with the planner, the planner can develop better plans by receiving information on the results of the plan. This will help avoid similar failure conditions in the future.

4.2 HIERARCHICAL PLANNING

4.2.1 METHOD

In hierarchical planning, some of the minor details of the problem are initially ignored and consideration is given to only the main issues. Once a solution can be found for the main issues, an attempt is made to fill in the appropriate details.

This can be achieved by giving the preconditions an abstract priority level. The first pass through the planning process will only consider preconditions at the highest priority level. After a plan has been created, consideration is given to preconditions at the next lower priority level. The plan is then augmented with those operators the satisfy the preconditions. This process is repeated until a complete plan is generated for all the different levels of preconditions.

Hierarchical planning can also be done using a plan graph. A plan graph is a tree that starts with a complete plan at the root node and continually subdivides the different components of the plan into subplans at the intermediate nodes. The base components of the plan are at the leaf nodes. The planner starting at either the root or the leafs of the tree builds a plan by traversing the tree. The plan consists of the subplans reached at the

different levels of the tree. The tree encompasses all possible partial orderings of the plan, decomposing the plan into every possible subplan.

4.2.2 EXAMPLES

ABSTRIPS (Nilson 1980) is the classic example of the first type of hierarchical planner. The hierarchical level of each precondition is simply indicated by a criticality value associated with each precondition. Small numbers indicate a low hierarchical level or small criticality, Large numbers indicate a high hierarchical level or large criticality. This is called a length-first search because the process explores the entire plan at one level of detail before it goes to the next lower-level.

AND/OR graphs (Homem de Mello and Sanderson 1986) is an example of the second type of hierarchical planning. In this system, an and/or graph is used to represent the assembly of a product made up of several different components. The initial state or leaf nodes, consist of all the components disconnected from one another. Intermediate states consist of different subassemblies of the components. The goal state or root node, is the complete assembly of the product. The robot can be presented with any combination of the components and by searching the graph, can decide what the best plan is for assembling the components.

Graphs of goal interactions can also be used to guide planning (Hayes 1987). In this case, graphs are used to model the behavior of a human machinist. An interaction graph is created to show the order in which the operations(drilling a hole, and milling) are performed on a piece of metal. A squaring graph describes the methods of squaring a piece of metal. Each graph shows how the operations are to be ordered for the different subgoals. The two graphs are then merged to generate a plan for the execution of both subgoals.

Each graph subdivides the goal, allowing for goal interactions to be identified when the merge operation is performed.

4.2.3 ADVANTAGES

When solving hard problems, the solution may consist of a very long plan. To construct large plans efficiently, it helps to concentrate on the basic problem first and to fill in the initially ignored details later.

By only considering the critical subgoals before the details, it can also reduce the search; by ignoring details one efficiently reduces the number of subgoals to be accomplished in any given sublevel. This reduces the amount of backtracking that has to be done because each subgoal's search space is small and therefore requires little backtracking in searching.

This contrasts with Resolution planning, which must search the entire world space which requires considerably more backtracking.

By traversing the space of all possible plans, graphs provide the opportunity to select an optimal plan and dynamically adapt planning to changing conditions in the environment. This is evident in (Homem de Mello and Sanderson 1986). If a robot is presented with two different parts to assemble that it does not normally expect, it is still able to find a plan to do the assembly.

An augmented RTN, described in the Goal Directed Example section (Georgeff and Bonollo 1983), is a simple graph inferencing mechanism that aids in both the acquisition of knowledge and in its verification. The fact that it is graphical in nature provides a very easy conceptual model for humans to understand; one can see the state transition arcs from one state to another, and why the transitions occur.

Plans and subplans are procedural by nature, and the construction of the planner is purely declarative. This permits easy construction with traditional languages, and is a big advantage for real-world planner implementations.

Procedural knowledge is knowledge stored as a sequence of operations to be performed. An example would be the knowledge built into a plan to move a robot's arm, which consists of "n" ordered executable steps.

Declarative knowledge is an explicit knowledge representation that can be interpreted as making declarative statements about the world. Such statements typically are stored in symbol structures that are accessed by the procedures that use this knowledge. For example, a declarative statement is ON(Block-A Block-B), which states Block A is on Block B.

4.2.4 DISADVANTAGES

The assignment of priority levels is critical to the success of Hierarchical planning. A detailed knowledge of these levels in its domain is required and may vary from situation to situation. When attempting to assign a priority one faces two problems: having to access the assignment in relation to the priorities assigned to all other preconditions, and of trying to anticipate if it will ever change in a new situation. For these reasons alone, hierarchical planning is not a good approach because it is very hard to manage these priorities in a large systems.

For hierarchical planning to be applicable, a plan must be specified from start to finish, but this is not possible in a dynamic environment. World conditions change, and the planning process should not have to start from the beginning again and again as the plan's environment changes. Some backtracking is indeed eliminated by hierarchical planners but the need to start at the beginning every time the environment changes goes beyond the usual disadvantage of excessive backtracking.

Graphs can be quite effectively used in small problems but become unmanageable in large systems. Moreover, since a state in the graph must be created for every part of the plan, this approach will not be feasible.

Graphs may lead to the construction of many correct plans. However, additional work must be done to ensure that the planner is using the most efficient one. A hierarchical planner could be always using the worst plan which is clearly not advisable in time critical planning applications.

4.2.5 MSS SUITABILITY

Hierarchical planning is planning at progressively greater levels of detail. This is not the type of planning we are attempting for the MSS. We know what the individual tasks are, and we just want to find out a way of completing all tasks with a given set of constraints. There is one master task which is the plan itself and a list of subtasks which must be achieved. These subtasks do not need to be broken down any further since they are already in an executable state. For this reason, and because of its inability to cope with environmental changes, Hierarchical planning is of little use to the MSS.

4.3 LEAST-COMMITMENT PLANNING

4.3.1 METHOD

Instead of considering all the permutations of an operator sequence, a least-commitment strategy chooses the order in which operators are performed. It tries to discover which operators are necessary for a plan to meet its goals, as well as any required orderings among them.

The planning process first decides what has to be done, regardless of the order, on the basis of the stated goals that are to be achieved by the planner. This part is typically a form of hierarchical planning. The planner begins by constructing an abstract skeleton of a plan and then, in successive steps, fills in more and more detail. By using hierarchies, goals are partitioned into subgoals for the refinement of the operators to meet the subgoals. Nothing is stated about the order in which the operators must be applied to meet the overall goals.

A second process then takes the operators required to meet the goals and orders them, if necessary, using their preconditions and constraints. The key point here is that a least-commitment planner does not bother ordering operators when this is of no concern to the planning process.

An example of this is planning to make breakfast. The goal of breakfast is to have toast, tea, and cereal. The operators required in this plan are Make(toast), Make(tea), Get(milk), and Get(cereal). To have breakfast, all four operations need to be done but not in any particular order. Therefore, the planner does not need to account for ordering, it must only find and execute the operators. However, there are cases where ordering is important. For example, in a four course meal one want to be sure the first course comes before the second, and so on, but the ordering within the courses may be independent of the ordering of the food for that course.

4.3.2 EXAMPLES

The prime example of this type of planning is NOAH (Cohen and Feigenbaum 1982), (Rich 1983). NOAH develops a plan from a procedural net. From a single node representing a goal or task to achieve, a hierarchy of nodes are created that represent a partially ordered set of tasks to achieve that goal. Similarly, each one of those subtasks has a partially ordered set of subtasks to achieve that subtask, etc. All tasks are linked together (by attaching subtasks to their parent task) to form a procedural net.

Thus, the original task is replaced by several layers of more detailed tasks located in its plan library. This hierarchy of tasks ends with immediately attainable tasks, executable by the plan's operators.

Interaction problems between operators are continually examined and corrected in the plan as they arise. This allows NOAH to solve interaction problems constructively; they are not ordered until a problem is detected and are ordered to prevent the problem.

4.3.3 ADVANTAGES

NOAH-like planners, by following the least commitment strategy, achieve more flexibility while doing less backtracking. By searching their plan library, plans for subtasks can be located and executed with little ordering conflict. Also, by having a plan built from partial plans, permutations of all possible operator sequences do not have to be considered for developing a plan.

By being partially a Hierarchical planner, Least-Commitment planners also benefit from that method's advantages.

4.3.4 DISADVANTAGES

Relying on plans that are in the plan library can be a problem in some cases. If the library does not contain the appropriate plan or subplan, the

planner is stuck since it cannot build a plan from scratch nor revise existing plans.

The process of recognizing interacting subproblems can become costly when the subplans go to extremely deep levels. If simple plans are used that are only four or five levels deep, this problem is of little concern. However, in deep subplans the complexity and time required to check all interactions of operations becomes a heavy strain on the system. Again, once the planning process has started this type of method can not deal with a changing environment. It requires the world to be in a stable condition. Plans are created from start to finish and are not permitted to be altered.

4.3.5 MSS SUITABILITY

Least-Commitment planners are not suitable for the MSS project for at least two reasons: they disregard the ordering of the tasks, and they are unable to continue planning if a ready-made plan does not exist in their plan library.

The MSS has deadlines or power resources limits that must be met, and a plan must be efficient enough to execute in the time - or with the resources - available. In such cases, the ordering of subtasks is crucial to

the effectiveness of the plan. Least-Commitment planners use a partial ordering and cannot detect conflicts in such situations.

If a Least-Commitment planner cannot find a plan in its library of ready-made plans, it cannot do anything else but give up. The MSS planner must always be able to achieve a plan and not have to give up because of lack of information.

4.4 CONSTRAINT POSTING PLANNING

4.4.1 METHOD

The previous least-commitment strategy can also be extended. In NOAH it was the order of operators we were concerned with, however, other decisions are faced by more complicated planning systems. In most planning systems, objects as well as operators are central to the planning process.

Constraint posting is a technique that allows decisions about objects to be deferred for as long as possible, even though these decisions may interact with one another. Constraints are relationships among plan variables. Using constraints, solutions to nearly independent subproblems can be found. For example, suppose you must elect two people, one for president and one for

treasurer. You do not want to consider all possible pairs of people, instead you want to break the problem into two nearly independent subproblems. The subproblems are not really independent since the two people must be able to work together. Let X be the president, Y be the treasurer, and the following constraints added to the requirements: COMPATIBLE(X,Y), and NOTEQUAL(X,Y). With these, the planning process can proceed as if working on two separate problems: finding a president, and finding a treasurer. Checks are done later to see if the proposed component solutions meet the constraints.

Constraints help refine general plans, that have many different solutions, to a few plans that have specific solutions. By using constraints, the planning activity can be more focused and able to continue planning where it might have otherwise had to stop. For example, if a vital piece of information is missing or not in support of the planning activity, such as the planner having only ten units of type X and it needs twenty, a constraint can be imposed, such as it will later receive ten more units of X, that will enable the planner to continue under the new constraint condition.

4.4.2 EXAMPLES

MOLGEN (Stefik 1981) is perhaps the best known planner using this technique. It uses three types of operations on constraints: constraint formulation, constraint propagation, and constraint satisfaction.

Constraint formulation is the adding of new constraints as the planning process proceeds, causing the problem to become more tightly specified.

Constraint propagation is the creation of new constraints from old constraints. When constraints are propagated, they bring together the requirements from separate parts of the problem, allowing a least-commitment strategy of deferring decisions for as long as possible.

The propagation of dominance relations (Wellman 1987) among classes of candidate plans is central to the planning framework of Wellman's system. By integrating a dominance prover into the plan search process, the traditional Constraint Posting planning method is generalized to permit partially satisfiable goals. Plan classes are generated by posting constraints at various levels of abstractions, then classified within a plan lattice that manages inheritance of properties and dominance characteristics.

Constraint satisfaction is the operation of finding values for the constraint variables so the set of constraints can be satisfied for a plan.

As well as dominance relationships, constraints can take the form of resources (Muscettola and Smith 1987) such as time, supplies, and equipment. In this paper, a probabilistic approach is presented to resource allocation and plan evaluation.

4.4.3 ADVANTAGES

The major advantage of constraint posting planning is its ability to anticipate interference between subproblems and to eliminate the interfering solutions as planning proceeds.

In addition, it shares the advantages of Least-Commitment planning. As constraints are introduced, the possible solution set is reduced, allowing for only viable solution paths to be checked and ruling out the rest. Constraints allow the solution set to be partitioned into different subproblems. Thus, the planner rarely commits itself to a decision it must undo later. This makes this method an efficient approach to guiding the plan to its correct solution.

Finally, this method is built on top of the hierarchical approach and gains all of its advantages.

4.4.4 DISADVANTAGES

Constraint posting is a knowledge intensive style of problem solving. It requires knowledge about how and when to generate a constraint and when to propagate it. One way of overcoming this disadvantage is to allow constraints to be specified only at the introduction of the planning process. But with this method, the dynamic flavor of constraints is lost. Another method would only allow the introduction of new constraints but not their propagations. This allows for the dynamic nature of an outside environment, while making the internal operation of the planner static.

4.4.5 MSS SUITABILITY

Constraint planning can play a major role in helping to solve the MSS problem. The following constraint types augment the MSS specific information available to the planner to help restrict the search space:

- Goal constraints - Provide details of why an action is in the plan, thereby, removing the need to deal with many apparent interactions and search alternatives.

- Object and Resource constraints - Provide the ability to recognize conflicting or concurrent use of shared objects and resources.

- Consumable Resource constraints - Restrict search and heuristically weight search alternatives to prefer low consumable resource use.

- Time constraints - Provide the ability to specify windows for goals or actions, and the ability to deal with external timed events. These constraints can also be used to restrict the search to solutions that can meet the given time specifications.

- Priority Constraints - Provide preferences to guide search.

4.5 GOAL DIRECTED / IMMEDIATE EXECUTION PLANNING

4.5.1 METHOD

As described previously, planning problems are posed with unique initial and final world states. In Goal Directed planning the problem is only specified by a goal. The planner does not commit to any specific course of events, but rather specifies appropriate reactions for anticipated situations. Another way to view this is that one large plan represents a solution to every other possible plan. Which part of the universal plan is to be executed depends on the current environment the plan is executing in.

Accordingly, this type of planning is called 'immediate execution planning'.

There is no commitment to any sequence of events, and the plan contains little sequence of any kind. Instead, the planner partitions the set of possible situations on the basis of the reaction each situation requires. The behavior of an agent executing the plan depends on which situation has arisen at execution time. This permits appropriate behavior even in unpredictable environments. It will execute what it thinks is correct for the given situation. Since it does not depend on past history, variable situations do not affect the current plan. A synthesizer of such plans determines the behavior by deciding under what conditions a feedback function should be invoked to achieve some change in the condition of the world. The feedback function retrieves information about the world so the planner can determine a response. It is of little use to retrieve information when the world has not changed.

This type of planning does not build and execute a complete plan, but rather interleaves the planning and execution of the plan. By anticipating every possible situation in a domain and prescribing an action for every initial state, the action is usually optimal.

4.5.2 EXAMPLES

Universal Plans (Schoppers 1987) is one example of an attempt to compactly represent every possible plan of the block world's "stack" operation, executable by a robot arm. The planner creates a decision tree for every possible world state that block A and B can be in and provides actions for each one of those states. Preconditions are used in the decision tree to help establish the current world's state. An action is taken by the planner when the world state matches the preconditions of a world state in the decision tree. The action is taken as an intermediate step in the completion of the goal of stacking two blocks. Execution of the partial plan provides for a new world state. The result of the subsequent execution of the partial plans should lead to the completion of the Universal Plan's goal of the "stack" operation.

Procedural Expert Systems (Georgeff and Bonollo 1983) use the concept of a recursive transition network (RTN) to guide and control the planner. An RTN is a network of nodes connected by arcs. An arc can be traversed from one node to another only if the predicate labeling the arc evaluates to true. Nodes correspond to the world states, and arcs correspond to preconditions and actions. All possible paths in an RTN node are explored, beginning at the initial arc and ending with the final arc. The arc predicates are any computable function and can include goal predicates used for the guidance of the planning process.

4.5.3 ADVANTAGES

By having a reactive planner, plans do not have to rely exclusively on what has happened in the past. This is an important feature when working in a dynamic environment, where world state descriptions can vary from the start to the finish of the execution of the plan. What is important is the goal of the plan, not the particulars taken to solve it. Replanning, due to changes in the parameters of the world, can be an expensive operation. A complete execution of a plan may never be completed in a dynamic environment. The early commitment problem is side-stepped by the interleaving of plan refinement and execution.

This method relies on its environment for information about the world after the execution of an action. Other planners must explicitly identify the changes in the environment using postconditions. In the real-world, actions do not always affect the world as they should or are expected to. Therefore, this type of interaction is hard to describe be means of postconditions used by other planners, but is easily handled by goal directed planning.

4.5.4 DISADVANTAGES

One disadvantage of this method is that the execution of an incomplete plan may make the goal permanently unachievable. The planner may bring about a world state that it can not get out of, like painting itself into a corner. The fact that it does not consider what it has done in the past leads to this fault. A true planner must consider where it was, where it is going, and what in the past is similar to its current situation. By using all of this information, the planner can make consistent plans from situation to situation.

Universal type plans should be limited to very small problem spaces. This is due to the amount of world states that can occur for any one problem, and the amount of information required for each state. In spite of its simplicity, a typical block world example has in excess of 400 possible world states. This makes this type of planner impractical for real-world spaces which are generally much larger.

The job of identifying every possible world state is a problem in itself especially when state descriptions and preconditions contain many different variables.

4.5.5 PART 8.5 - MSS SUITABILITY

This type of reactive planner can be of use when the MSS plan is executing. If the plan fails for some reason, lack of resources, this type of planning can be used to suggest ways that the plan can continue executing. The MSS is a two part planner, with a planning and execution stage. The planning phase uses predictions about resource allocation, which may not always be correct. Therefore, in the execution phase it is essential to have a re-planner that can dynamically react to situations that cause the current plan to fail. Relying on reactive planning and a replanning capabilities saves time that would otherwise be wasted generating a new plan from scratch, and may enable the planner to salvage the partial plan by making use of the currently available resources and information.

4.6 CASE-BASED PLANNING

4.6.1 METHOD

Planners such as STRIPS and NOAH try to create every plan without using, and benefiting from, any past experiences in solving a problem that is similar to the current needs. Case-Based planning attempts to reuse existing

plans, either directly from a plan library if the current situation is similar to a pervious situation, or by altering past plans to meet the current situation.

Most of human planning uses this model. Over time we encode our plans into memory and try to fit them to our current environment. It is rare that one experiences a situation that one has not already encountered or that does not resemble a past experience. However, occasionally a new situation is encountered and one must use other methods of planning. After the new plan is generated, one has a plan that can be used over and over in similar situations in the future.

One component of Case-Based planning is the plan library. As the name suggests, this is a library of proven plans that have been used in the past. Each plan contains a ordered sequence of actions to perform and a list of goals that the plan satisfies. Because the planner must be able to reuse old plans and store new ones, it must be able to understand and explain why a plan has succeeded or failed in a given situation. This information is used in the future to make a better plan. The source of failure information is also used as an index to the plan. This means that the planner must have a powerful memory organization as well as a strong model of causality of the domain in which it operates.

Another component of Case-Based planning is the plan retriever. Given the currently defined goals a plan must achieve, the plan retriever looks into the plan library and attempts to locate a plan that satisfies those goals. If it can not find one, it will then attempt to find one that is similar to

the current situation. In the Example section, we describe a planner called PLEXUS (Alterman 1986). It provides an approach to identify similar plans to the current situation.

After a plan has been found, it is passed to the plan modification component. This component will alter the plan, if required, to match the current situation. This component uses domain information to help modify the plan. For example, if a retrieved plan satisfies a goal not required by the current plan and the goal is achieved by some action in the plan, that action can be removed from the plan without causing any harm to the completion of the plan's other goals.

Next, the plan is passed to the execution component of the planner. After a plan has been executed, the results are used to decide what to do with the plan. If the plan succeeded, the plan is added to the plan library together with the goals it satisfied. If it failed, the plan is analyzed to detect the failure component of the plan. The failure information is added to the existing plan in the library so it can be used to avoid the same problem again in the future.

4.6.2 EXAMPLES

A good example of this type of planning is CHEF (Hammond 1986). CHEF is a Case-Based planner which creates and learns new plans in the domain of

Szechwan cooking. CHEF also attempts to do a little more than the standard Case-Based planner. It tries to anticipate planning problems by noticing features in the current input that have previously contributed to planning problems. It also attempts to do plan repair when a plan has failed. In addition to noting exceptions, it tries to fix them and store the newly debugged plan.

PLEXUS (Alterman 1986) goes into more detail about how plans different from the current situation are matched and corrected for use in the current planning situation.

PLEXUS identifies four ways that plans can differ from a situation. The first way is when a failing precondition differs from the current situation. In this case, PLEXUS tries to generalize the precondition to see if it can find an alternative precondition it can use. The second way is when the failing outcome(goal) differs. In this case, the desired outcome of the current situation differs from the outcome of the stored plan. The planner then attempts to modify the plan so the failing condition will not occur. The third way is when a goal differs. Incremental abstraction or specialization is done in an attempt to change the goal. The final way plans can differ is when situations are out of order. The planner can either, on the basis of domain knowledge, reorder some steps, or remove them completely from the plan.

Planning with Abstractions (Tenenberg 1986) describes how abstraction hierarchies of domain specific information can be used to help adapt old plans to new ones. The planner determines if an old plan can be appropriately

altered to fit a new plan by adapting the situation, goal, or object within the new plan, using the abstraction hierarchies.

WOK (Hammond 1983) demonstrates the use of goals as the main index to old plans. It suggests an organization and indexing strategy that allows the retrieval and use of plans that are composed of sets of goals rather than just individual goal situations.

An important, yet frequently studied, part of Case-Based planning is a method of locating a plan in a very large search space. POISE (Carver, Lesser, and McCue 1984) implements a search focusing mechanism that checks the syntactic and semantic validity of a possible plan. It also uses a heuristic focus-of-control by providing a context mechanism. This allows interpretations which were considered unlikely if newly discovered information validates them.

4.6.3 ADVANTAGES

The biggest advantage of the method of Case-Based planning is not having to start the planning process from scratch. By using existing plans, a very efficient planning architecture can be developed. One wants a planner that can learn and reuse complex plans, rather than a planner that must always start from scratch to achieve a plan for the same set of goals.

This type of planner can also use the information of a failed plan to anticipate and avoid similar failures in future sessions. The anticipate-and-avoidance mechanism is one advantage other planners do not provide, and is vital to the success of a planner.

The planner also has the ability to produce many different types of plans. By using different criteria of similarity, many past plans may be applicable to the current situation. This provides a diverse choice of plans.

4.6.4 DISADVANTAGES

The biggest problem of Case-Based planning is identifying a plan that is similar to the current situation. The search can not be too tight, otherwise no plan will be located. If, on the other hand, the search is too loose, plans that are irrelevant to the current situation will be found. The problem is to use the correct blend of searching strategies.

If the plan library is large, better search techniques will be needed to focus in on the correct plans. In particular, one does not want to spend a long time searching for a plan that does not exist.

Another problem occurs when no plan can be located in the plan library. If no plan exists, the planner must produce a new plan by one of the methods

that uses primitive actions to plan from scratch. If this interface is not provided, the planner will be doomed to fail in an automatic mode.

Finally, the planner has the problem of trying to decide whether it is worth while to alter an existing plan, or should it use one of the planning-from-scratch methods. Additionally, after the plan is altered, executed, and is ready to be stored in a library, how does the planner know the alteration has been successful? These decisions must be made correctly to support plan learning.

4.6.5 MSS SUITABILITY

This is the most suitable of the existing types of planners for the MSS problem. The ability to use old plans is central to any dynamic planning approach. Indexing old plans, using their goals, and using the problems they helped to avoid, can provide one with complex plans without going through the complex plan development process. We can also use old plans to develop new plans when a ready-made one is not available. By locating old plans that are similar to the new situation, we can use the differences to develop the new plans. The MSS is not an application that needs to reason about planning tasks because the tasks are well defined. However, it must reason about how to order these tasks, and this is where references to past plans and their ordering of tasks can be of great value.

5.0 CURRENT SCHEDULING SYSTEMS

The following is a survey of NASA-related and other expert systems designed for scheduling. The systems' names and short descriptions are provided. A more detailed discussion can be found in (Liebowitz and Lightfoot 1987). Development of the systems were done by NASA, Universities, and private industries.

1. DEVISOR - An expert system designed for the scheduling of deep-space missions.

2. KNEECAP - A planning aid for constraint-based crew activity scheduling.

3. PLAN-IT - An expert system for schedule planning.

4. EMPRESS - Stands for Expert Mission Planning and Replanning Scheduling System. Aids in constructing and maintaining particular mission schedules.

5. CAPS - Stands for Crew Activity Planning System. Is an expert system that is a planning generator at the Johnston Space Center.

6. IEPS - Stands for Interactive Experimenter Planning System. Is a generalized expert system scheduling tool to aid satillite experimenters in their request generation.

7. RPMS - Stands for Resource Planning and Management System. Assist users with general planning and scheduling tasks.

8. EXEPS - Is an expert system for timelining electrical power system activity blocks.

9. BATTLE - Allocates weapons to targets for battle management purposes.

10. ISA - Stands for Intelligent Scheduling System. Is an expert system to schedule orders for manufacturing and delivery.

11. KNOBS - Performs mission planning.

12. ISIS - Is an expert system to schedule manufacturing steps in job shops.

13. NUDGE - Is a framed based expert system used in scheduling.

Our preliminary design approach (Deugo, Oppacher, and Thomas 1988) draws on the best design concepts and lessons learned from these previous systems. By using them, we can gain from their experiences and avoid their failures. These concepts are identified in the preliminary design approach paper.

6.0 OUR PROPOSED APPROACH

Planning is an important part of a person's day to day activities. A plan can be both informal or formal by nature. It is used to meet a person's goals using the resources he has around him. Businesses run their operations arranging, coordinating, and planning resource allocations to achieve an objective. We see then that the MSS problem is one that is being encountered over and over again in the world, and is not one that is only specific to the MSS. All of these plans must use resources such as time, labor, capital, machinery, and information, attempting to maximize the efficiency with which they are consumed.

Typical schedulers, not based on AI techniques, have a proven record of obtaining optimum solutions when the supply and demands of resources can be specifically identified. However, when the environment is only informally specified, and when the specifications are possibly incomplete, inconsistent, and variable, AI techniques may lead to a big improvement over traditional schedulers because they provide a more adaptive planning mechanism.

Our approach to the MSS planning problem consists of four components: the plan identifier, the plan executor, the dynamic replanner, and the plan evaluator.

The plan identifier component uses a dynamic memory approach to generate a plan. First, the user provides the list of tasks and constraints to set up

the plan identification phase. Using this information, the planner indexes into a library of old plans, indexed by their goals and failures, and attempts to find a plan that matches the current situation. If a plan can be found, that plan is passed to the plan execution component. If no plan can be found, the planner will attempt to find one that is similar to the current situation. The criteria for partial or similar matching is outlined in the planner design document (Deugo, Oppacher, and Thomas 1988). This plan is massaged by the component's domain plan information, planning heuristics, and operator input, if so desired, to create a new plan used by the plan execution component.

This approach enables the planner to continue planning even though the planner has no ready-made plan to deal with the current situation. This feature make the planner an advancement over the current planners that must always start planning from scratch, or are only looking for plans that exactly match the current situation.

The plan execution component takes the plan and starts the execution of it. If the plan experiences a failure condition(eg. lack of resource), the exception is noted and the dynamic replanning component is activated. This component will attempt to reorder the plan, remove failing tasks, stop execution, or ask the operator for assistance in order to keep the plan's execution continuing. When a plan fails, one does not want to stop the execution because resources have been allocated and are ready to use. The dynamic replanning component is another important advancement in planning, since it prevents minor faults from stopping plan execution but causes only a slight replanning of the plan. The replanning component uses the plan-

ning technique known as goal planning or reactive planning to keep a plan executing.

After the plan has executed, the plan, its exceptions, and its replanning descriptions are given to the plan evaluator. If the plan was an old plan that executed successfully, this information is added to the plan in the plan library to give added support information to it. If the plan was a newly created one and it executed successfully, the plan is added to the plan library along with the goals it satisfied. If the plan failed, the exceptions along with why the goals failed are noted. If the plan has a bad track record it may be altered using the replaning information to make it a better plan. The user is part of this activity and helps verify the reasoning of the planner and to ensure the sanity of the plan library updates.

This component helps the planner acquire new plans and knowledge by learning from itself, both in creating new plans and from its planning failures. It also learns to fix plans and adapt to new environments over time.

All the components make use of dynamic memory; to generate a plan, to execute and alter a plan, and to store and update new and old plans. By using dynamic memory, the goal of providing the MSS with some degree of autonomy is achieved. The MSS can work in a dynamic environment and be ready to meet the changes it will face. To achieve a robust, dynamic, and efficient planner for the MSS, we feel capabilities of handling resource constraints, feedback, and operator input, that use dynamic memory planning techniques are critical.

Following this document is the preliminary MSS design document. This document provides an in-depth look at the approach that has been outlined here. It includes an MSS planner synopsis, description, and outline of the design that will be used for the implementation phase of the MSS.

7.0 REFERENCES

1. Allen, J. F., A general Model of Actions and Time, Computer Science Report TR 97, University of Rochester, Rochester, New York, 1981.

2. Alterman, R., An Adaptive Planner, pages 65-69, AAAI, 1986.

3. Carbonell, J. G., Learning by Analogy: Formulating and Generalization Plans from Past Experience, pages 137-162, Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann Publishers, 1983.

4. Carver, N. F., Lesser, V. R., and McCue, D. L., Focusing in Plan Recognition, pages 42-48, AAAI, 1984.

5. Cohen, P. R, and Feigenbaum, E. A., The Handbook of Artificial Intelligence, pages 555-562, William Kaufmann, 1982.

6. Deugo, D. L., Oppacher, F., Thomas, D., A Proposed Approach for Scheduling Applications (With Respect to The Mobile Servicing System),1988.

7. Georgeff, Michael P., and Bonollo, Umberto, Procedureal Expert Systems, pages 151-157, IJCAI, 1983.

8. Georgeff, Michael P., Lansky, Amy, L., and Bessiere, Pierre, A Procedureal Logic, pages 516-523, IJCAI, 1985.

9. Hammond, K. J., Planning and Goal Interaction, pages 148-151, AAAI, 1983.

10. Hammond, K. J., CHEF: A Model of Case-Based Planning, pages 276-271, AAAI, 1986.

11. Hayes, Caroline, Using Goal Interactions to Guide Planning, pages 224-228, AAAI, 1987.

12. Hayes, P. J., The Frame Problem in Artificial Intelligence, pages 223-230, Readings in Artificial Intelligence, Morgan Kaufmann Publishers, 1981.

13. Homem de Mello, L. S., and Sanderson A. C., And/Or Graph Representations of Assembly Plans, pages 1113-1119, AAAI, 1986.

14. Kolodner, J. L., Simpson, R. L., and Sycara-Cyranski K. L., A Process Model of Case-Based Reasoning in Problem Solving, pages 284-290,IJCAI 1985.

15. Liebowitz, J., and Lightfoot, P., Expert Scheduling Systems: Survey and Preliminary Design Concepts, pages 261-283, Applied Artificial Intelligence, V 1, 1987.

16. Manna, Z., Waldinger, R., A Theory of Plans, pages 11-45, Reasoning About Actions and Plans, Morgan Kaufmann Publishers,1987.

17. McDermott, D., A Temporal Logic for Reasoning about Plans and Processes, Computer Science Research Report 196, Yale University, New Haven, Connectict, 1981.

18. Muscettola, N., and Smith, S. F., A Probabilistic Framework for Resource-Constrainted Multi-Agent Planning, pages 1063-1066, IJCAI, 1987.

19. Nilson, Nils J., Principles of Artificial Intelligence, pages 275-360, Palo Alto, California Tioga Press, 1980.

20. Nilson, Nils J., and Genesereth M. J., Logical Foundations of Artificial Intelligence, pages 263-306, Morgan Kaufmann Publishers, 1987.

21. Rich, Elaine., Artificial Intelligence, pages 247-294, Mcgraw-Hill Book Company, 1983.

22. Sandewall, E., and Ronnquist, R., A Representation of Action Structures, pages 89 - 97, AAAI, 1986.

23. Schoppers, M. J., Universal Plans for Reactive Robots in Unpredictable Environments, pages 1039-1046, IJCAI, 1987.

24. Stefik, M., Planning with Constraints(MOLGEN: Part1), pages 111-140, Artificial Intelligence, V 16, # 2, May 1981.

25. Tenenberg, J., Planning With Abstraction, pages 76-80, AAAI, 1986.

26. Wellman, M. P., Dominance and Subsumption in Constraint-Posting Planning, pages 884-890, IJCAI, 1987.

27. Winslett, Marianne, Validating Generalized Plans in the Presence of Incomplete Information, pages 261-266, AAAI, 1987.