MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES

3 4456 0329039 2

ORNL/TM-11308
(CESAR-89/34)

ornl

OAK RIDGE
NATIONAL
LABORATORY

MARTIN MARIETTA

Job Planning and Execution Monitoring for a Human-Robot Symbiotic System

Lynne E. Parker

OAK RIDGE NATIONAL LABORATORY
CENTRAL RESEARCH LIBRARY
CIRCULATION SECTION
4500N ROOM 175
LIBRARY LOAN COPY
DO NOT TRANSFER TO ANOTHER PERSON
If you wish someone else to see this report, send in name with report and the library will arrange a loan.

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.
NTIS price codes—Printed Copy: A04 Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-11308
CESAR-89/34

Engineering Physics and Mathematics Division

JOB PLANNING AND EXECUTION MONITORING FOR A HUMAN-ROBOT SYMBIOTIC SYSTEM

Lynne E. Parker

Center for Engineering Systems Advanced Research

Date Published - November 1989

Prepared for the
Engineering Research Program
Office of Basic Energy Sciences
U.S. Department of Energy

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
operated by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES

3 4456 0329039 2

TABLE OF CONTENTS

Section	Page
ABSTRACT	vii
1.0 INTRODUCTION	1
2.0 JOB PLANNING FOR A HUMAN-ROBOT SYMBIONT	3
2.1 INTRODUCTION	3
2.2 JOB PLANNER INPUT	4
2.2.1 Action Operator Rules	4
2.2.2 Goal Condition	5
2.2.3 Environmental Conditions	6
2.2.4 Object Types	6
2.2.5 Condition Difficulties	7
2.3 JOB PLANNING ALGORITHM	7
2.3.1 Determining When the Top Goal is True	8
2.3.2 Applying ADD/DELETE LIST Conditions	9
2.3.3 Building the Execution Monitoring Table	9
2.3.4 Backtracking	10
2.3.5 Finding an Operator with an ADD LIST Condition matching the Current Goal	10
2.3.6 Selecting the "Best" Rule to Use When More Than One is Available	10
2.3.7 Instantiating Parameters	11
2.3.8 Creating the DTA Task Tree	11
2.4 AN IMPLEMENTATION OF THE JOB PLANNER	12
3.0 EXECUTION MONITORING FOR A HUMAN-ROBOT SYMBIONT	17
3.1 INTRODUCTION	17
3.2 DETECTING PROBLEMS IN TASK EXECUTION	17
3.2.1 Using the Execution Monitoring Table	17
3.2.2 Using Expected Execution Time	20
3.2.3 Updating the Environmental World Model	21
3.3 REPLANNING DUE TO TASK EXECUTION PROBLEMS	21
3.4 AN IMPLEMENTATION OF THE AUTOMATED MONITOR	22
4.0 JOB PLANNER AND AUTOMATED MONITOR INTERFACE TO OTHER SYMBIONT MODULES	25
5.0 CONCLUSIONS AND FUTURE WORK	27
6.0 REFERENCES	31
APPENDIX A -- EXAMPLE ACTION OPERATOR RULES	A-1
APPENDIX B -- EXAMPLE JOB PLAN FOR CRANFIELD ASSEMBLY	B-1
APPENDIX C -- EXAMPLE EXECUTION MONITORING TABLE	C-1

LIST OF FIGURES

Figure		Page
1	Human-Robot Symbiont Architecture	2
2	Sample Task Tree	13
3	Cranfield Benchmark in Disassembled State	13
4	Assembled Cranfield Benchmark	14
5	Job Planner Program Flow	28
6	Dynamic Task Allocator Program Flow	29
7	Automated Monitor Program Flow	30

ABSTRACT

The human-robot symbiosis concept has the fundamental objective of bridging the gap between fully human-controlled and fully autonomous systems to achieve true human-robot cooperative control and intelligence. Such a system would allow improved speed, accuracy, and efficiency of task execution, while retaining the human in the loop for innovative reasoning and decision-making. Earlier research has resulted in the development of a robotic system architecture facilitating the symbiotic integration of teleoperative and automated modes of task execution. This architecture reflects a unique blend of many disciplines of artificial intelligence into a working system, including job or mission planning, dynamic task allocation, human-robot communication, automated monitoring, and machine learning. This report focuses on two elements of this architecture: the Job Planner and the Automated Monitor.

1.0 INTRODUCTION

During the last few decades, a growing awareness and belief has arisen that automation-related technologies and intelligent machines will play an increasing role in improving the development and operation of complex and advanced systems. In this context, research and development have taken place on a broad range of technologies aimed at achieving advanced systems varying from fully remotely-controlled systems, such as advanced teleoperators and servomanipulators, to fully autonomous intelligent robots involving artificial intelligence, super-computing, machine vision, and advanced control. Within this large spectrum of technological research, work has been initiated on a new class of automated systems which offers promise for improving the productivity, quality, and safety of operation of advanced systems. This new type of automated system is referred to as "Human-Robot Symbiosis."

In a symbiotic system, humans and robots cooperate in the decision-making and control of tasks in a complex, dynamic environment, communicating frequently in the exchange of tasks. The fundamental objective of human-robot symbiosis is to bridge the gap between autonomous and human-controlled systems by merging the advantages of fully autonomous systems (e.g. efficient repetitive task execution and immunity from fatigue) with those of fully human-controlled systems (e.g. expertise in a wide variety of task domains and the ability to cope with unexpected events). The function of the symbiotic system is to dynamically optimize the division of work between the human and the robot, with the ultimate goal of improving the admissible task range, accuracy, and work efficiency of the system. The successful creation of such systems requires an effective approach to several fundamental technical issues, such as human-robot communication, autonomous task planning and execution monitoring, dynamic task allocation, human-robot system architecture, and machine learning via experience and human observation. (Refer to [8] for more details on human-machine symbiosis.)

Earlier research has resulted in the development of a robotic system architecture facilitating the symbiotic integration of teleopertive and automated modes of task execution [9]. Shown in Fig. 1, the architecture reflects a unique blend of many disciplines of artificial intelligence into a working system. Previous research has also resulted in the development of a methodology for Dynamic Task Allocation, described in [10-12]. This report focuses on the methodologies developed for two additional elements of the symbiont architecture: the Job Planner, described in Section 2, and the Automated Monitor, described in Section 3. Section 4 describes the interface of these modules to the other components of the symbiont architecture, while concluding remarks are given in Section 5.

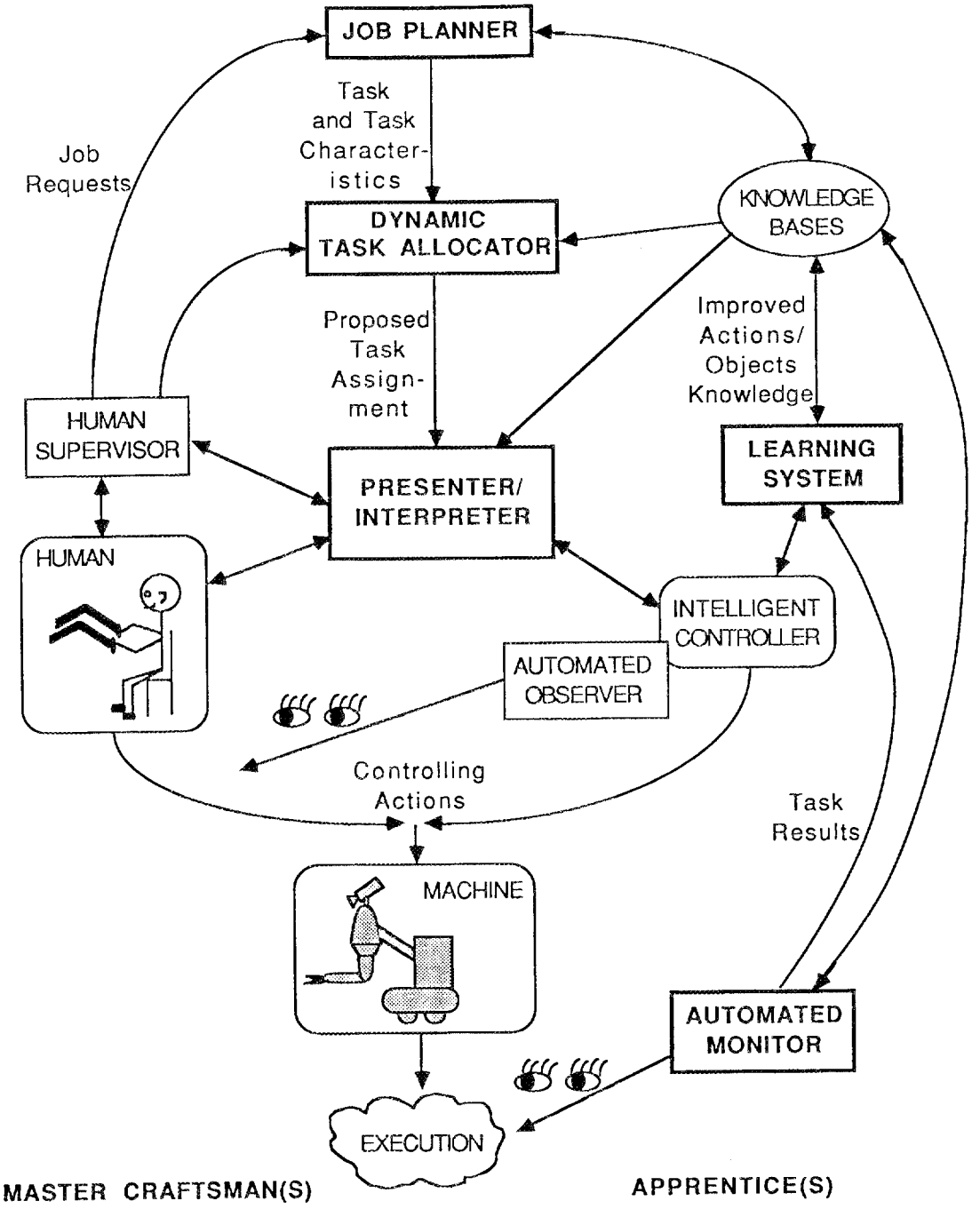

Fig. 1. Human-Robot Symbiont Architecture.

2.0 JOB PLANNING FOR A HUMAN-ROBOT SYMBIONT

2.1 INTRODUCTION

In a human-robot symbiotic system, the Job Planner is responsible for planning the primitive task activity sequences that lead to efficient job completion. The task strategy derived by the Job Planner is important, as it establishes the steps to be followed by the human and the robot to complete the job, or achieve the goal. Many job planning methodologies currently exist which provide various approaches to the job planning problem [for example, 3-5, 13-17]. In this symbiotic system, the selection of a job planning approach was closely related to the definition of a language through which the modules of the symbiont communicate. This language provides the basis for the interaction of the modules during symbiont operation, and consists of an action-object relationship. A set of valid actions which can be performed in the environment is defined, along with the set of objects which exist in the symbiotic world and the relationships between the actions and objects. These sets will likely vary over time as new actions are learned, or as new objects appear in the world. Each of the components of the symbiont architecture uses this language to accomplish its objectives: the Job Planner plans actions to be performed on objects, the Dynamic Task Allocator assigns actions to be performed to the human or the robot, the Automated Monitor observes the execution of actions or the states of objects, and the Learning System learns new actions or object characteristics.

With this action-object language in mind, the job planning methodology used by the symbiotic system was chosen to exploit the relationship between the actions to be performed and the objects in the task execution environment. This methodology utilizes an initial starting state, a goal state, and a set of action operator rules to generate a sequence of operators which transforms the planning world model from the initial state to the goal state. The Planner performs a depth-first search of the action operators to create a linear, non-hierarchical plan. The initial and goal states consist of modified first-order predicate calculus statements defining conditions which describe the world state. Ideally, this model would precisely reflect the state of the true world during actual plan execution. An operator is a description of an action which may be performed by the human or the robot, and has a syntax similar to that of the STRIPS planning system [3], consisting of the set of conditions which must be true before the action operator can be exercised and the conditions which either become true or become false subsequent to the application of the operator.

The job decomposition derived by the Job Planner is passed to the Dynamic Task Allocator in the form of a tree for the assignment of tasks to the human or the robot. The task sequence derived by the Job Planner must allow for rapid reconfiguration due to problems in task execution or changes in the human or robot capabilities. This reconfiguration will occur in close cooperation with the Automated Monitor, described in Section 3, which is responsible for detecting events requiring modification of the task plan.

The following sections describe the Job Planning methodology in more detail.

2.2 JOB PLANNER INPUT

In order to plan jobs, the Job Planner must be provided with the following input data:

1) Action operator rules to use during planning
2) Initial environmental conditions, or planning world model
3) Goal condition
4) Information on the "types" of the objects in the environment (required when the action rules include references to object types, indicated by prefixes of "#" in object references. For example, "Small_pin" would be the type of particular pins named "Small_pin_1" and "Small_pin_2".)
5) Condition difficulties, giving the conditions which are considered to be the most difficult to accomplish (required if more than one rule can be used to achieve the same effect)

The Job Planner expects data items 1-3 to be provided in a particular format created exclusively for this purpose. This format can be easily expressed as production rules in Backus-Naur form. The following notations are used for these expressions:

lower case	indicates syntactic categories
::=	"to be written as" symbol
|	vertical bar separating choices (OR)
$\{\ \}_{1}$	choose 1 of the enclosed items
$\{\ \}_{1+}$	repeat the enclosed items 1 or more times
$\{\ \}_{0+}$	repeat the enclosed items 0 or more times
$\{\ \}_{opt}$	optional items
other items are	terminal symbols of the language

The subsequent sections describe the required input data in more detail.

2.2.1 Action Operator Rules

The syntax for the action operator rules is similar to the STRIPS format, and must be expressed according to the following productions:

operator_rules ::= $\{$operator_decl$\}_{1+}$

operator_decl ::= operator_name PRECONDITIONS: $\{$condition$\}_{0+}$
END
DELETE_LIST: $\{$condition$\}_{0+}$
END
ADD_LIST: $\{$condition$\}_{0+}$
END

operator_name ::= identifier { ({parameter}$_1$ {,parameter}$_{0+}$) }$_{0+}$

condition ::= identifier { ({parameter}$_1$ {,parameter}$_{0+}$) }$_{0+}$

parameter ::= { * | # }$_{opt}$ identifier | -

identifier ::= {alphachar}$_{1+}$

alphachar ::= ! | % | & | ' | + | - | . | / | 0 | ... | 9 | : | ; | < | +
| > | ? | @ | A | ... | Z | [| \ |] | _ | a | ... | z | '|'

(i.e., any printable character expect 'space', 'comma', #, {, }, *)

(NOTE: Four types of parameters are possible with the above definitions. An identifier preceded with "*" refers to an uninstantiated parameter (refer to Section 2.3.7 for more information on parameter instantiation) which can be matched to anything, as long as it is instantiated to the same value in every occurrence of the parameter in an operator rule. For example, "*object" in "Grasped(*object)" could be instantiated to "Grasped(Casing)" or "Grasped(Lever)." An identifier preceded with "#" refers to an uninstantiated parameter which must be matched to a value (i.e. object) of a certain type (given by the identifier) throughout the rule. For instance, the condition "Grasped(#Bolt)" could be instantiated to "Grasped(Bolt_1)" or "Grasped(Bolt_2)," but not "Grasped(Lever)." An identifier without a prefix must remain exactly as is given in the rule. For example, "Curr_loc" in "Move_Arm(Curr_loc,*to_loc)" must remain as "Curr_loc". Finally, a parameter of "-" serves as a type of place-holder, meaning that it can match anything. An example of this type is given by "At(Hand,-)", in which the "-" could be matched to anything, such as is shown in "At(Hand,Lever:Hover_pos)" or "At(Hand,Starting_loc)").

An example of a valid set of action operator rules used to plan the Assembly/Disassembly of the Cranfield Benchmark (which is discussed in Section 2.4) is included in Appendix A.

2.2.2 Goal Condition

The goal condition provided by the human through the human-machine interface (Presenter/Interpreter) must be in the following format:

goal_condition ::= condition

In the context of the Cranfield Benchmark and using the sample rules given in Appendix A, the input goal for a Cranfield assembly would be "Assembled(Benchmark)," while the goal for a disassembly would be "Disassembled(Benchmark)."

2.2.3 Environmental Conditions

The environmental conditions describing the initial state of the world must be in a file in the following format:

environmental_state ::= {condition}$_{1+}$

For example, if the Cranfield Benchmark is in a disassembled state and the robot end-effector is empty at the beginning of task execution, the environment could be described as follows:

```
At(Hand,Starting_Loc)
Handempty
Positioned(Casing1,Casing1_home)
Positioned(Casing2,Casing2_home)
Positioned(Lever,Lever_home)
Positioned(Spacer,Spacer_home)
Inserted(Peg,Peg_home)
Inserted(Large_pin_1,L_pin1_home)
Inserted(Large_pin_2,L_pin2_home)
Inserted(Large_pin_3,L_pin3_home)
Inserted(Large_pin_4,L_pin4_home)
Inserted(Small_pin_1,S_pin1_home)
Inserted(Small_pin_2,S_pin2_home)
Inserted(Small_pin_3,S_pin3_home)
Inserted(Small_pin_4,S_pin4_home)
Inserted(Small_pin_5,S_pin5_home)
Inserted(Small_pin_6,S_pin6_home)
Inserted(Small_pin_7,S_pin7_home)
Inserted(Small_pin_8,S_pin8_home)
```

These conditions compose the initial planning world model, sometimes referred to as the environmental model, which the Job Planner will update during planning to reflect the environmental effects of actions added to the job plan.

2.2.4 Object Types

The information describing the object types is obtained from the objects knowledge base and is necessary to allow the Job Planner to handle operator rules which refer to generic types of objects, rather than specific objects. For example, an operator rule can include a precondition that a small pin is to be grasped ("Grasped(#S_pin)") without naming an individual small pin. This allows more than one object to satisfy the required conditions, thus providing the possibility for more flexible job plans. When the Job Planner encounters a parameter preceded by '#', it cannot instantiate the parameter without first comparing the type of the possible instantiating value with the required parameter type. The parameter can only be instantiated if the object types match.

2.2.5 Condition Difficulties

A file containing a list of the conditions that are more "difficult" to achieve must be provided if more than one operator rule can be used to realize the same effect. This situation would occur in planning applications that allow more than one set of actions to be used to reach the goal. The intent of the operator rule structure is for the job planner to derive a plan most suitable for the initial starting environment, which will usually be different for each job start. Depending on the application, several levels of condition difficulties can be established, resulting in a hierarchy of difficulties. For example, a "Grasp" action could be considered to be more difficult than a "Find" action, but less difficult than an "Insert" action, in general. The current implementation only allows a hierarchy of two levels and requires that the difficult conditions be contained in a file in the following format:

difficult_conditions ::= $\{\text{identifier}\}_{0+}$

Refer to Section 2.3.6 for more details on the use of this data.

2.3 JOB PLANNING ALGORITHM

Once the Job Planner has been furnished with the required input data, it can derive a plan to reach the goal. In terms of the previously described input data, the job planning methodology can be expressed as follows: let $E = \{e1, e2, \ldots\}$ be the set of initial environmental conditions, g be the goal condition, and $T = \{t1, \ldots, tm\}$ be the set of m action operator rules, where each ti consists of 3 lists:

$P = \{p_{i1}, p_{i2}, \ldots, p_{i\,maxp}\}$, list of preconditions
$A = \{a_{i1}, a_{i2}, \ldots, a_{i\,maxa}\}$, add conditions
$D = \{d_{i1}, d_{i2}, \ldots, d_{i\,maxd}\}$, delete conditions,

where each $p_{ih}, a_{ij}, d_{ik} \in E$, $1 \le i \le m$, $1 \le h \le maxp$, $1 \le j \le maxa$, $1 \le k \le maxd$.

The Job Planner's task, then, is to find the sequence of actions $t_1, \ldots, t_n$, such that condition g is true subsequent to execution of t_n. The algorithm used by the Job Planning system to implement this methodology is described below in an outline form, followed by additional details of the planning procedures.

A. Initialize; receive the goal from the human and push it onto an empty goal stack.

B. While the goal stack is not empty, do the following:
   1. If the top goal is true (see Section 2.3.1), do the following:
      a. If the top goal is the last goal needed for an operator to be applied, do the following:

1). Verify that all of the other preconditions of this operator are still true.
   a) If so, do the following:
      i) Apply the ADD and DELETE lists of the current operator to the current planning model (see Section 2.3.2).
      ii) Build the EMT entry for this operator for the Automated Monitor (see Section 2.3.3).
      iii) Add the current operator to the job plan.
      iv) Pop the top goal off the goal stack.
   b) If not, backtrack (see Section 2.3.4).

b. Otherwise (i.e., the top goal is not the last goal needed for an operator to be applied):

   Pop the top goal off the goal stack.

2. Otherwise (i.e., the top goal is not true)
   a. Find all the operators which have an ADD LIST condition matching the top goal which is to be met (see Section 2.3.5).

   b. If no operators are found, backtrack (see Section 2.3.4).

   c. Otherwise (i.e., operators were found), if more than one operator rule was found, select the "best" rule to use (see Section 2.3.6).

   d. Instantiate the parameters of the selected rule to those required by the top goal condition (see Section 2.3.7).

   e. Push the preconditions of the instantiated operator rule onto the goal stack.

C. If a job plan was found, create the task tree for the Dynamic Task Allocator (see Section 2.3.8);

   otherwise, signal that a plan could not be found.

The current planning algorithm ends the search with the first-obtained plan that reaches the goal (or with a message that a plan could not be found), without attempting to find an alternative plan. Although this method is not ideal, further research must be undertaken to ascertain how to distinguish the "goodness" of various alternatives. A Job Planner which could derive plans that are optimal in terms of logic (the most "sensible" plan), cost (the plan with the fewest number of operators), or time (the plan quickest to execute) would be much more powerful.

2.3.1 Determining When the Top Goal is True

The goal condition at the top of the goal stack is found to be true in the following ways:

a) The goal condition matches a planning world model condition exactly.

b) The goal condition has an uninstantiated parameter (denoted by the first character being '*') which, if instantiated to a parameter of an existing environmental condition, will make the goal true (e.g., the goal "At(Hand,*curr_loc)" will match the environmental condition "At(Hand,Pin1:Grasp_pos)" if "*curr_loc" is instantiated to "Pin1:Grasp_pos").

c) The goal condition requires an object of a certain type (denoted by the first character being '#'), and a condition exists with an instantiated parameter of this type (e.g., the goal "Grasped(#S_pin)" will match the environmental condition "Grasped(S_pin_1)").

If the above process determines that the top goal must be instantiated to particular values to be true, ALL of the parameters of the corresponding operator rule must be instantiated consistently with the values required by the current goal condition. Refer to Section 2.3.7 for more details.

2.3.2 Applying ADD/DELETE LIST Conditions

Applying the ADD LIST conditions is done simply by adding the ADD LIST conditions to the list of conditions which describe the current planning world model. The DELETE LIST conditions are applied by searching the current planning world model for a match of each of the DELETE LIST conditions. All matching conditions are deleted from the world model. A "match" can either be an exact match, or a situation in which the DELETE LIST condition type (e.g. Grasped, Placed, etc.) matches an environmental model condition type, and the unmatching DELETE LIST parameters equal '-' (meaning "anything"). For example, the DELETE LIST condition "At(Hand,-)" matches the environmental condition "At(Hand,S_pin_1:Hover_pos)". Thus, the "At(Hand,S_pin_1:Hover_pos)" condition would be deleted from the environmental model.

2.3.3 Building the Execution Monitoring Table

The Execution Monitoring Table (EMT) is built as the job is planned. As each new operator is added to the job plan, an EMT entry for that operator is created. The entry consists of the instantiated preconditions of the new operator, a set of continuing conditions, and the instantiated ADD and DELETE lists of the current condition. The continuing conditions are those conditions which were on the ADD LISTs of earlier operators but have not yet appeared on the PRECONDITIONS list of a subsequent operator.

Once the job is planned, the Execution Monitoring Table is available to the Automated Monitor for use during job execution to detect problems in task execution. Refer to Section 3.2 for more details on the use of the Execution Monitoring Table.

2.3.4 Backtracking

Backtracking will be required when, during the course of accomplishing one or more of the preconditions of an operator, one or more of the previously-met preconditions of that operator was (were) "un-done" --- i.e., it (they) became false. An alternative use of the operators to achieve the preconditions must be found. To handle this type of scenario, more sophisticated planning techniques must be utilized, such as the use of critics in the NOAH planning system [13], or the use of constraint posting in the MOLGEN System [14, 15]. The backtracking technique must be sufficiently intelligent to recognize previously attempted planning paths to avoid infinite searches for a job plan, or cycling; without such intelligence, the job planning algorithm is not guaranteed to terminate. A backtracking technique has not been implemented in the current job planning system.

2.3.5 Finding an Operator with an ADD LIST Condition Matching the Current Goal

A match of the top goal on the stack with an operator ADD LIST condition is found when any of the following are true:

a) The top goal matches an ADD LIST condition of an operator rule exactly.

b) The type of the top goal condition (e.g. Grasped, Handempty, At, etc.) is the same as an ADD LIST condition of an operator, and the non-matching parameters in the ADD LIST condition are uninstantiated (i.e., the parameter begins with '*'). For example, the goal "Grasped(Casing1)" will match the ADD LIST condition "Grasped(*object)" if "*object" is instantiated to "Casing1."

c) The type of the top goal is the same as an ADD LIST condition of an operator rule and the non-matching parameters in the ADD LIST condition are uninstantiated objects of a certain type (i.e., the parameter begins with '#').

2.3.6 Selecting the "Best" Rule to Use When More Than One is Available

Depending upon the application, action operator rules can be designed to allow more than one operator to be used to achieve a desired condition (for example, there may be more than one way to perform an assembly task). In these situations, a method must be incorporated for selecting the

"best" rule to use. This selection is accomplished using two heuristic measurements: percent-achieved (PA) and percent-difficult (PD).

The percent-achieved factor gives the percentage of the rule's preconditions that are currently met in the planning environmental model. The percent-difficult factor gives the percentage of the currently-met preconditions which are "difficult" to achieve. As explained in Section 2.2.5, this factor is based upon the concept that some conditions are "easier" to be met than others. From a set of applicable operators, the one with the highest PD factor is selected; however, if the rules have the same PD factor, the selected rule is that with the highest PA factor. If the rules still tie, the rule found first is arbitrarily selected.

2.3.7 Instantiating Parameters

In this report, the term "instantiate" refers to the process of binding a variable (parameter) to a specific value. For instance, instantiation of the variable "*object" to "Spacer" in the condition "Grasped(*object)" would result in "Grasped(Spacer)." It is very important that when a parameter of a rule is instantiated, all other instances of that parameter in the rule are instantiated equally for that particular use of the rule.

When instantiating the parameters of a condition or rule, it is useful to be able to instantiate only a portion of a parameter, rather than the entire parameter. For example, consider the following rule:

```
Place(*object,*loc):  PRECONDITIONS:  Grasped(*object)
                                      Found(*loc)
                                      At(Hand,*loc>hover_pos)
                                      END
                      DELETE_LIST:  END
                      ADD_LIST:  Positioned(*object,*loc)
                                 END
```

in which "*loc" is to be instantiated to "Site_1." The Job Planner will replace only the exact character string "*loc" with "Site_1," rather than the entire parameter, resulting in the following (notice the "At" condition):

```
Place(*object,Site_1):  PRECONDITIONS:  Grasped(*object)
                                        Found(Site_1)
                                        At(Hand,Site_1>hover_pos)
                                        END
                        DELETE_LIST:  END
                        ADD_LIST:  Positioned(*object,Site_1)
                                   END
```

In this example, the "At" condition became "At(Hand,Site_1>hover_pos)" rather than "At(Hand,Site_1)." An additional instantiation of "*object" to, for example, "Lever" would complete the instantiation of this rule.

2.3.8 Creating the DTA Task Tree

This step is necessary to convert the derived job plan into a format useful for the Dynamic Task Allocator (DTA). The DTA expects the job plan in the form of a task tree that gives a hierarchical breakdown of the job to be performed down to the elemental sub-subtask level. The Job Planner currently plans at the subtask level --- that is, the operators are equivalent to subtasks in DTA terminology. Since these operators (subtasks) are not the most elemental actions that the human and the robot can perform (for example, the robot operates at the lowest level in joint rotations), additional research should be performed to determine the breakdown of the actions to the most primitive level. In the meantime, the Job Planner provides this information to the Dynamic Task Allocator by equating the subtask information with the elemental sub-subtask data. The only exception to this is in the case of the FIND and MOVE_ARM actions, which, in the current implementation, must be performed by the same resource and not allocated separately. This is due to the current limitation of human-robot communication which does not facilitate rapid communication of object locations between the human and the robot. The FIND and MOVE_ARM actions thus become elemental sub-subtasks of a common subtask, which will then be allocated as one unit by the Dynamic Task Allocator. Logically, then, a sample task tree for a portion of the Cranfield assembly appears as shown in Fig. 2. The Job Planner converts its plan to this tree format and writes the results to a file for the Dynamic Task Allocator. Refer to the Dynamic Task Allocation documentation in [10-12] for more details on the task tree format and the Dynamic Task Allocation methodology.

2.4 AN IMPLEMENTATION OF THE JOB PLANNER

The job planning algorithm has been implemented using the "C" programming language, first on an IBM-PC, and then on an OS-9 68020 system. A demonstration scenario was selected to illustrate the results of the symbiont architecture, including the job planning and execution monitoring methodologies. The task, called the "Cranfield Benchmark Assembly/Disassembly," represents fundamental characteristics of practical small component manipulation tasks, and was developed as a European benchmark by the Cranfield Robotics and Automation Group at the Cranfield Institute of Technology in Bedfordshire, England [2]. Figures 3 and 4 illustrate the Benchmark in the disassembled and assembled states. As shown in these photos, the Cranfield Benchmark consists of 17 parts --- 2 casings, 1 lever, 1 spacer, 1 peg, 4 large pins, and 8 small pins. The actions required to assemble/disassemble the Benchmark include Grasp, Release, Insert, Place, Move_Arm, and Find.

After being provided with the action operator rules listed in Appendix A, the initial environmental conditions given in Section 2.2.3, and a goal condition of "Assembled(Benchmark)," the Job Planner derived a job plan containing 119 actions, as shown in Appendix B. Note that the additional inputs of object types and condition difficulties discussed in Section 2.2.4 and 2.2.5 were not required, since the action operator

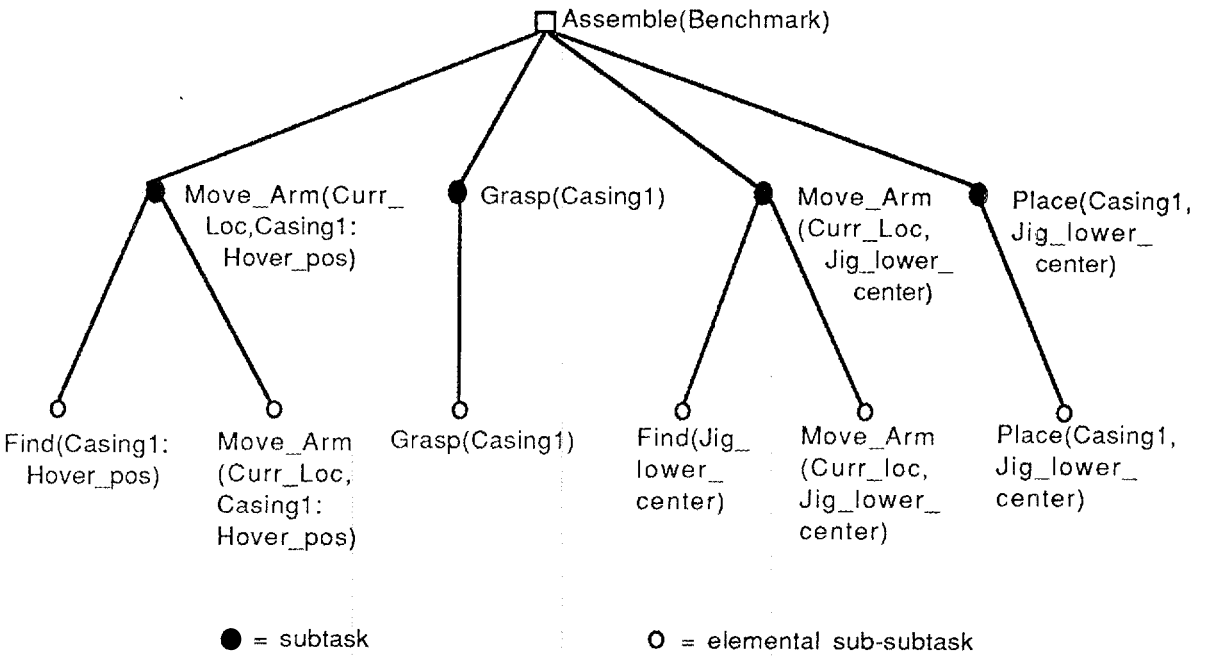

Fig. 2. Sample Task Tree.

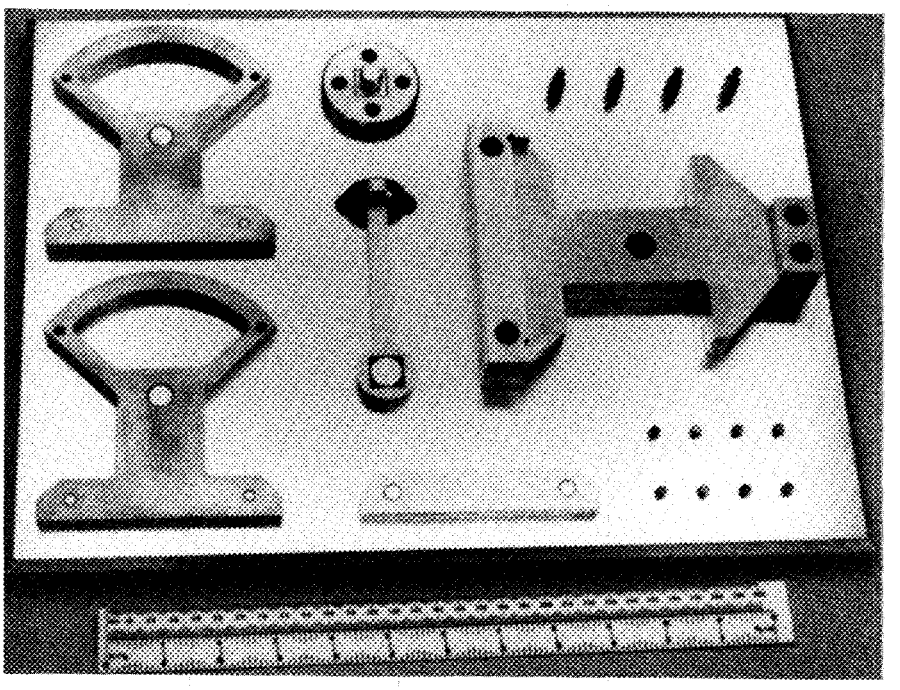

Fig. 3. Cranfield Benchmark in Disassembled State.

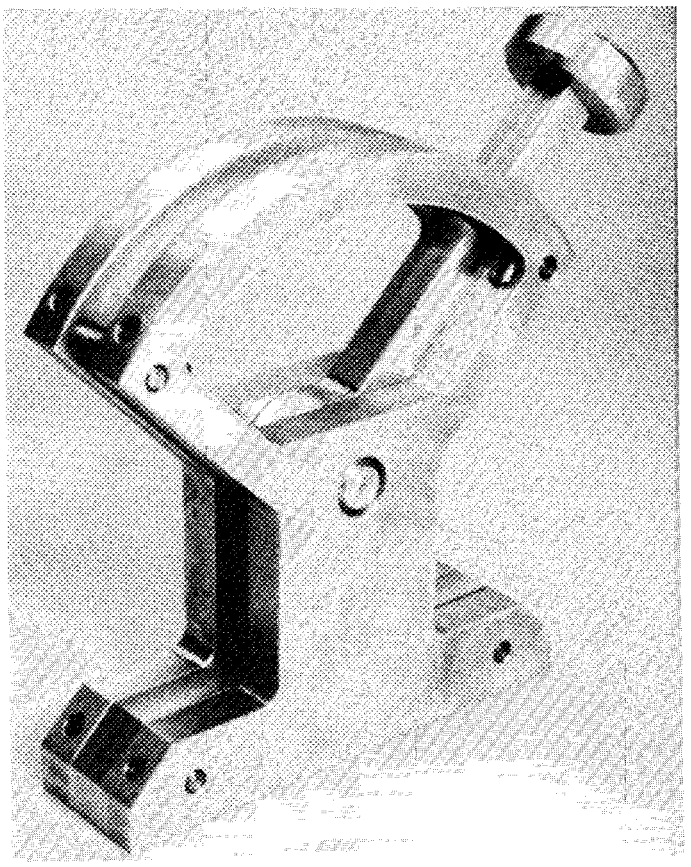

Fig. 4. Assembled Cranfield Benchmark.

rules in this example do not use preceded with "#", and no two can result in the same effect.

The processing time required to plan this job on an OS-9 68020-based system was approximately 20 "wall clock" seconds. The increase in processing time with increasing job plan length for this sample set of operator rules is linear, requiring approximately one second for every six actions in the plan. Of course, the planning time requirements for the general case will be dependent upon the number of operator rules utilized and their complexity. Although not instantaneous, the typical processing time should be quite acceptable for the applications envisioned for this type of planning system.

3.0 EXECUTION MONITORING FOR A HUMAN-ROBOT SYMBIONT

3.1 INTRODUCTION

In the human-robot symbiont, the Automated Monitor module fulfills the execution monitoring responsibilities during task performance. In this role, several key functions are provided by the Automated Monitor, all of which revolve around the observation of the task execution and the selection of appropriate actions to be taken based upon the observations. First of all, the Automated Monitor is responsible for detecting when problems in task execution have occurred -- a particularly important function since the environment is assumed to be hostile. This aspect of the Automated Monitor focuses on the importance of sensing and the integration of the planning and sensing operations. After determining that a problem has occurred, the Automated Monitor proceeds to perform its second function -- working to derive a modified job plan to reach the goal in spite of the task execution problem. The philosophy behind the approach to this responsibility is to re-use the existing job plan to the fullest extent possible to prevent a needless waste of time in re-planning. The ability to recover from execution errors is a key to improving the performance of the symbiotic system and to the reduction of the fragility of the system. The third duty of the Automated Monitor is to maintain the consistency between the real world and the knowledge bases. This function is of particular importance to the computerized elements of the symbiont, since they rely on the accuracy of the knowledge bases to perform their tasks. The final responsibility of the Automated Monitor is to observe the actual performances of the resources (human and robot) during task execution and adjust the resource capability parameters as necessary to reflect the true resource capabilities. This function is significant for the Dynamic Task Allocator, since it uses the capability parameters to derive appropriate task allocation recommendations.

Although all four of the Automated Monitor functions are important, this report focuses on the first two: detecting problems in task execution and recovering from those problems. The following sections provide more detail on these functions.

3.2 DETECTING PROBLEMS IN TASK EXECUTION

3.2.1 Using Execution Monitoring Table

A number of sensor-based execution monitoring strategies have been investigated in the literature (for example, [1,6]). While several of these approaches would apply to symbiotic systems, the monitoring approach developed for this project was selected to correlate with the job planning methodology described in the earlier section. This correlation is crucial, since the Job Planner possesses the knowledge required to successfully monitor job execution. This section describes the methodology used for the detection of problems in task execution.

How adeptly the Automated Monitor can detect that a problem has occurred is directly related to sensor feedback; not only must sensor

feedback be available to the Monitor, but the sensors must also be measuring parameters that provide insight to the process being monitored. For example, manipulation-related conditions, such as "Grasped(*object)", "Handempty," or "At(Hand,*loc)," can be verified using sensors such as force/torque, gripper width, tactile, and end-effector position sensors, whereas object-relationship conditions, such as "At(Wrench,Toolbox)," "On(Box1,Box2)," or "Connected(Bolt1,Plate4)," can be verified using vision sensors. The basic concept is to compare the current (usually, preprocessed) sensor readings with the expected sensor readings at critical points in time to determine if any inconsistency is detected. If so, the human in the human-robot symbiotic system is informed of the inconsistency to verify that a problem exists, and is requested to intervene in the task execution if necessary.

To accomplish this, the Automated Monitor uses the following methodology:

Define:

Available sensor suite:

$S = \{s_1, s_2, \ldots, s_n\}$, where n = total number of sensors

Generic types of conditions describing the environment:

$C = \{c_1, c_2, \ldots, c_m\}$, where m = number of types of conditions

Actual sensor readings at a given point in time:

$R = \{r_1, r_2, \ldots, r_n\}$

In addition, for all c_i, let $V^1{}_i$, $V^2{}_i$, ... , equal the sets of valid sensor readings expected for each of the n sensors for condition c_i, where:

$$V^1{}_i = \{v^1{}_{i1}, v^1{}_{i2}, \ldots, v^1{}_{in}\}$$
$$V^2{}_i = \{v^2{}_{i1}, v^2{}_{i2}, \ldots, v^2{}_{in}\}$$

Then, for an expected condition c_i, if $r_j = v_{ij}$, $j = 1, \ldots, n$ for some set V_i, no problem exists, and execution can proceed normally. Otherwise, a discrepancy has been detected and the human will be notified of the potential problem.

As an example, assume that the available sensor suite, S, equals {tactile, gripper_width, position}, and that the generic conditions describing the environment, C, are {Grasped(*object), Handempty, At(Hand,*loc)}. Further assume that the tactile sensor returns either a 0 or a 1, meaning "no object sensed" or "object sensed," respectively; the gripper_width sensor returns a non-negative number indicating the width of the gripper (0 implies closed); and that the position sensor returns the three position coordinates (x, y, z) and three orientation angles of the end-effector. Then, for each condition in C, the sets of expected sensor readings V_i can be derived as follows:

for c_1 = Grasped(*object),

$$V^1{}_1 = \{ 1, >0, - \}$$

for c_2 = Handempty,

$$V^1{}_2 = \{ 0, -, - \}$$
$$V^2{}_2 = \{ -, 0, - \}$$

for c_3 = At(Hand,*loc)

$$V^1{}_3 = \{ -, -, *loc \}$$

(where '-' means the reading is not applicable, or does not matter)

Now, assume that at a certain point during task execution, the Automated Monitor expects the condition "Grasped(*object)" to be true. It obtains the actual sensor readings and finds that:

R = { tactile_reading = 1,
gripper_width = 2.5,
position = (3,4,5,90,45,60) }

Comparing these readings to the expected readings, $V^1{}_1$, it determines that 1 = 1 (tactile), and 2.5 > 0 (gripper_width), so the "Grasped" condition appears to be true. One modification to $V^1{}_1$ could be to make the expected gripper_width equal the actual width of the object being grasped. This would allow verification that the correct object is being grasped, rather than just any object.

The two sets of valid expected sensor readings, $V^1{}_2$ and $V^2{}_2$, for c_2, "Handempty," correspond to the gripper being either open or closed with no held object. In these situations, if the tactile sensor detects nothing, the gripper sensor reading is immaterial, whereas, if the gripper width is zero, the tactile reading is unnecessary. In either case, the position sensor is not applicable.

The information the Automated Monitor uses to make these comparisons comes directly from the information on the conditions that must be true at any given point in time. The question thus becomes -- How does the Automated Monitor know what conditions must be true at any given point in time? At first glance, it might appear that the information would come from the current action being performed. However, consider the action of moving the robot arm from point 'a' to point 'b'. The action is the same regardless of whether or not an object is being grasped during the move, but the sensor readings are not the same. Instead, what is required is detailed knowledge about the overall plan to be executed, not just the current action.

Since the Job Planner knows the overall intent of the plan to be executed (i.e., it knows the goal and subgoals), it can provide information to the Automated Monitor on what conditions should be true at any point during the execution of the plan. This information is relayed to the Automated Monitor in the form of an Execution Monitoring Table (EMT), which includes the expected conditions both prior to subtask execution (preconditions) and during subtask execution (continuing conditions).

Recall that the Job Planner produces a list of actions, $t_1, \ldots, t_n$, such that the goal condition g is true subsequent to the execution of action tn. The EMT, then, consists of the following for each task tk, $1 \leq k \leq n$:

PRECONDITIONS: Preconditions, P_k, of action t_k
CONTINUING CONDITIONS: All ADD LIST conditions, A, of previous tasks $t_1, \ldots, t_{k-1}$, which have not yet appeared on the PRECONDITIONS list, P, or DELETE LIST, D, of a subsequent task.
ADD LIST CONDITIONS: ADD LIST conditions, A, of action t_k
DELETE LIST CONDITIONS: DELETE LIST conditions, D, of action t_k

The intuition behind the composition of the set of continuing conditions is that the effects of tasks in a job plan should remain true until they are needed as preconditions of a later task, or until they are nullified by a later task. Future enhancements to this rule could 1) distinguish between desired effects of operators and side-effects (which are not needed by any future task), and 2) allow conditions to remain in the set of continuing conditions when needed by more than one subsequent task. The use of the ADD and DELETE LIST conditions is described in Section 3.2.3.

Using this table, the Automated Monitor is thus able to compare the actual sensor readings to the expected sensor readings both prior to and during task execution to detect problems in task execution. Once a problem event has been detected during task execution, the Automated Monitor works with the human to resolve the problem, since the human can more easily determine the significance of a detected problem. First, the human could instruct the Monitor to simply ignore the problem and continue with task execution. Presumably, the supposed problem is actually a false alarm, so no corrective action needs to be taken. Secondly, the human can instruct the Monitor to reallocate the task. In this situation, the human believes that the current subtask can be performed, but that the other resource (human or robot) would have a better chance of successfully completing the subtask. In this situation, the Monitor would inform the Dynamic Task Allocator of the human's request, and the Allocator would re-allocate the current subtask. Task execution would start anew while the newly-assigned resource attempted the subtask. The final option available for the human is to instruct the Automated Monitor to replan the job. In this circumstance, the human realizes that the current job plan is not sufficient to correct the problem, and a new or revised plan must be derived. Section 3.3 provides details on this replanning strategy.

3.2.2 Using Expected Execution Time

In addition to the knowledge available in the Execution Monitoring Table, the Automated Monitor can also obtain knowledge from the Dynamic Task Allocator concerning the expected subtask completion time. Typically, this information will be used during the performance of a job that was allocated based upon a policy of minimizing the expected

execution time (see [10-12]). During subtask execution, the Monitor can use this information to detect unexpectedly long attempts to complete a subtask and warn the human of suspected problems. This feature is of particular assistance during robot performance, since the human might not be aware of the reasonable length of time required for the robot to perform the current subtask. The human can respond by either instructing the Automated Monitor to ignore the elapsed time, re-allocate the subtask, or replan the job.

3.2.3 Updating the Environmental World Model

As described in Section 2, the Job Planner relies on the accuracy of the current environmental model to derive feasible job plans. Obviously, if the world model is not consistent with the actual environment, the derived job plan will probably be incorrect. The important step of maintaining the accuracy of the world model is performed by the Automated Monitor during job execution. The information required to achieve this function is provided to the Automated Monitor via the Execution Monitoring Table, which, along with the pre- and continuing condition for each operator, contains the instantiated ADD and DELETE LIST conditions of each action operator in the plan. Subsequent to the successful completion of each task, the Automated Monitor updates the environmental model with these ADD and DELETE LIST conditions of the action, thus maintaining the accuracy of the world model for future use by the Job Planner.

3.3 REPLANNING DUE TO TASK EXECUTION PROBLEMS

After detecting that a problem has occurred and receiving instruction from the human to replan, the Automated Monitor works to recover from the error by deriving a new or revised plan to achieve the goal. Ideally, the existing job plan should be re-used to the fullest extent possible to prevent a waste of time in re-planning. Only those portions of the plan which are no longer useful in reaching the goal should be replaced with a new sequence of operators.

A number of recovery techniques are possible in this scenario, varying from doing nothing to undergoing complete replanning (refer to [7] for a review of existing techniques). The replanning technique developed for this symbiont system involves utilizing pre-defined recovery strategies for the task execution problems detectable with the available sensor suite, and undergoing complete replanning when such a recovery strategy is unavailable. This strategy was considered to be more appropriate for the symbiosis project than the development of an elaborate recovery strategy, since more elaborate strategies could probably not be used with the limited sensing capabilities of the symbiont.

Formally, the replanning strategy can be stated as follows:

Let	$P = \{p_1, p_2, \ldots, p_j\}$	the set of j problem events that the Automated Monitor is able to detect and recover from
	$FX = \{F_1, F_2, \ldots, F_j\}$	the set of plan fixes for each event in P

where:

$F_i = \{t_{i0}, t_{i1}, \ldots\}$	the list of subtasks to be instantiated, then inserted into the plan to recover from the problem event i

Then, when a problem event $p_k \in P$ occurs, the subtasks in set $F_k \in FX$ are instantiated appropriately and inserted into the job plan prior to the subtask during which the event occurred.

As an example, assume that the Automated Monitor is able to detect an object being dropped. Thus, the set P = {dropped(*object)}. Further assume that during the execution of a "Move_Arm" action, the Monitor detects that an object, say a "Lever," has been dropped -- that is, "Grasped(Lever)" is no longer true. The Automated Monitor consults the human, who instructs the Monitor to replan the job. This causes the Monitor to look up the plan fix for a dropped object, F_1, which consists of the following tasks:

```
F1 = { Find(*object)
       Move_Arm(*object:Hover_pos)
       Grasp(*object) }
```

The Monitor must determine the identity of the dropped object by extracting the object's name from the "Grasped" condition (in this case, "Grasped(Lever)") of the Execution Monitoring Table condition which became untrue. The plan fix, F_1, is then instantiated to become:

```
F1 = { Find(Lever)
       Move_Arm(Lever:Hover_pos)
       Grasp(Lever)   }
```

These tasks are then inserted into the job plan prior to the Move_Arm action that was being performed, and are given to the Dynamic Task Allocator for allocation to the human or the robot. Once human approval is obtained, task execution begins again to re-grasp the Lever. If the problem event had not been a member of set P, the Monitor would have instructed the Job Planner to undergo complete replanning to correct the problem, since a simple fix is unavailable.

3.4 AN IMPLEMENTATION OF THE AUTOMATED MONITOR

This methodology has been implemented in "C", first on an IBM-PC, and then on an OS-9 68020-based system, and tested with the Cranfield

Benchmark scenario discussed in Section 2.4. A portion of the Execution Monitoring Table generated by the Job Planner and used by the Automated Monitor is given in Appendix C. In the demonstration set-up, the sensors available to the Automated Monitor were the tactile, gripper-width, force-torque, and end-effector position sensors. With these sensors, the Automated Monitor could monitor the conditions "Grasped," "At(Hand,*loc)," and "Handempty," thus allowing the detection of problem events such as "object dropped" or "object not grasped as expected." The Automated Monitor was also able to monitor the elapsed execution time to warn the human when task execution time was exceeding the expected time. The Automated Monitor successfully detected these problem events with simulated sensors during a task execution simulation and worked with the human to resolve the problems, using the replanning strategy when requested.

4.0 JOB PLANNER AND AUTOMATED MONITOR INTERFACE TO OTHER SYMBIONT MODULES

Within the architecture developed to illustrate human-machine symbiosis, it is crucial that all of the reasoning modules work together for the sharing of knowledge and the transfer of control. This characteristic is particularly important during task execution when problem events, task failures, or even normal task completions occur. Each of the modules must be able to determine the current state of the symbiont. Synchronized program flows were developed and implemented for the Job Planner, Dynamic Task Allocator, and Automated Monitor to allow the knowledge-sharing and transfer of control to occur. These program flows are shown in Figs. 5, 6, and 7. Inter-module communication was accomplished by the sending and receiving of messages between modules, implemented in the OS-9 system as pipes. With these program flows and through the use of messages, the high-level reasoning modules (Job Planner, Dynamic Task Allocator, and Automated Monitor) successfully cooperated in the simulation of a symbiont execution which included problems in subtask execution.

5.0 CONCLUSIONS AND FUTURE WORK

Methodological approaches to job planning and execution monitoring in a human-robot symbiotic system have been presented and discussed. These approaches have been shown to be successful for planning and monitoring a simulated task execution containing problem events. Continuing work should proceed toward interfacing these modules with real sensors during task execution on actual hardware to allow further testing of the planning and monitoring strategies. Continued enhancement of the planning methodology should concentrate on the development of hierarchical and optimal job plans, along with planning using incomplete information. Automated Monitoring capabilities can be enhanced with the availability of more sophisticated sensors, such as vision. The capability to analyze and interpret such complex sensors for use during job execution would greatly expand the execution monitoring potential.

Job Planner Program Flow

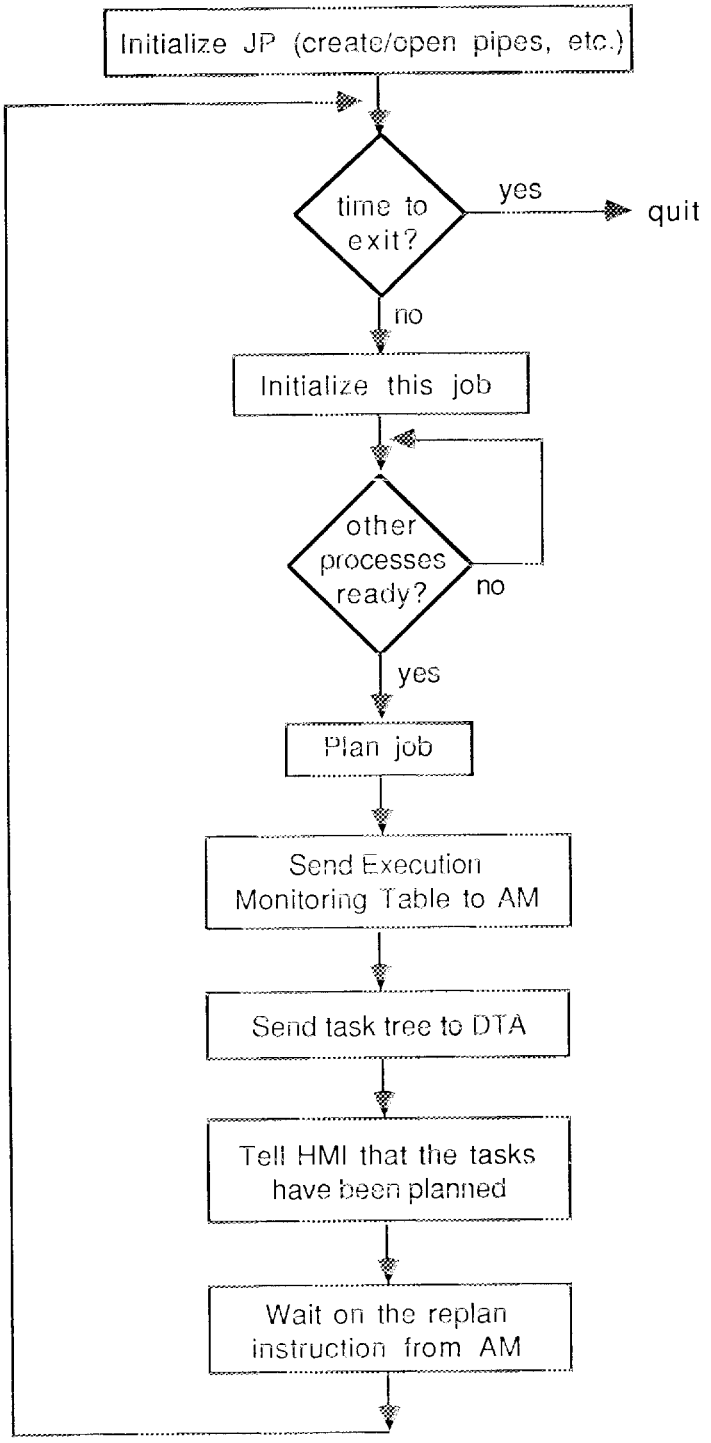

Fig. 5. Job Planner Program Flow.

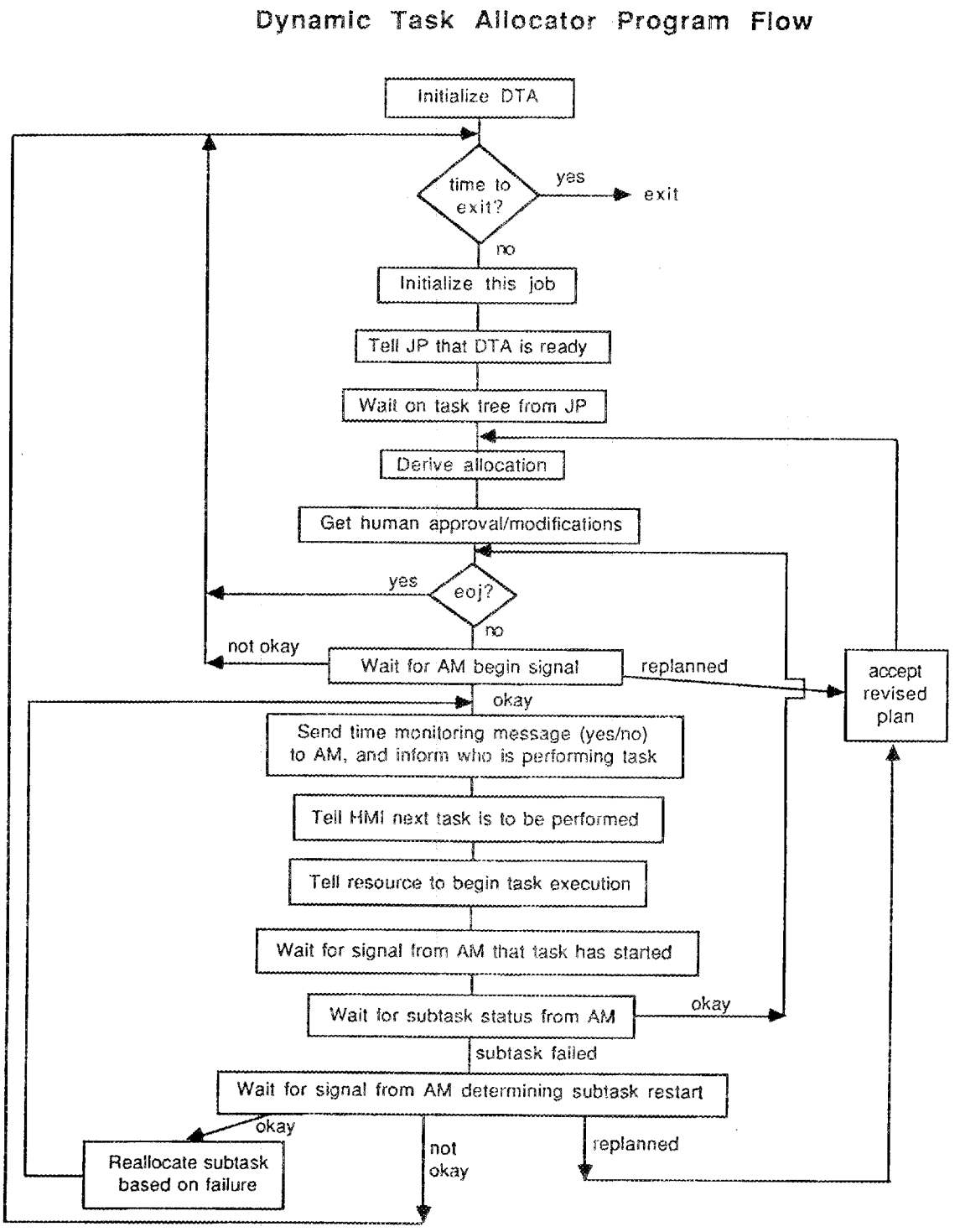

Fig. 6. Dynamic Task Allocator Program Flow.

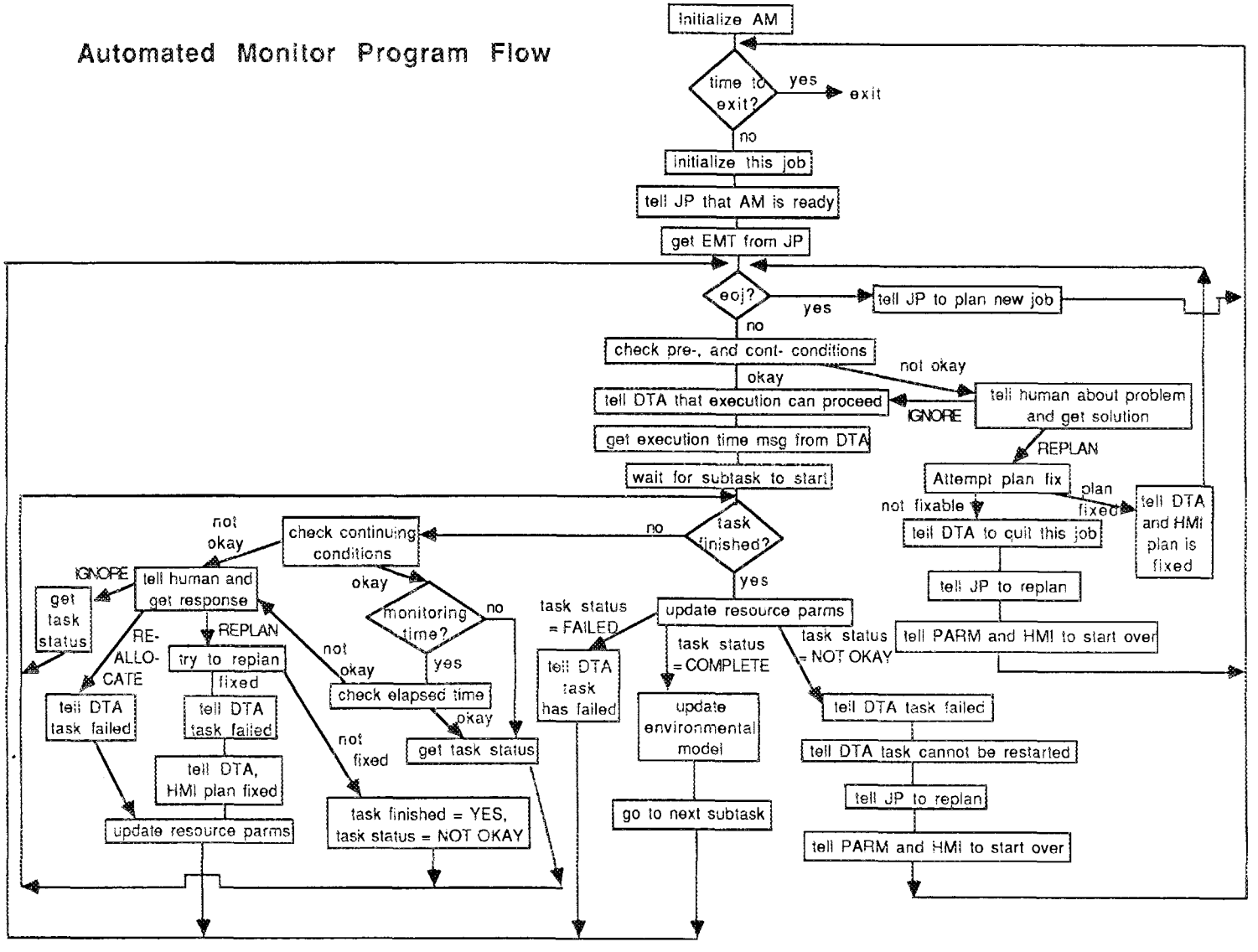

Fig. 7. Automated Monitor Program Flow.

6.0 REFERENCES

1. Chern, M-Y. "An Efficient Scheme for Monitoring Sensory Conditions in Robot Systems," Proc. of IEEE Int. Conf. on Robotics, Atlanta, 298-303, (1984).

2. Collins, K., A. M. Palmer, K. Rathmill, "The Development of a European Benchmark for the Comparison of Assembly Robot Programming Systems," Proceedings, June 1984 Robots Europe Conference, Brussels, Springer-Verlag, (1985).

3. Fikes, R. E. and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence 2, 189-208 (1971).

4. Fikes, R. E., P. E. Hart, and N. J. Nilsson, "Learning and Executing Generalized Robot Plans", Artificial Intelligence 3, 251-288 (1972).

5. Fikes, R. E., "Monitored Execution of Robot Plans Produced by STRIPS," Proc. IFIP Congress 71 (1971).

6. Gini, M., "Symbolic and Qualitative Reasoning for Error Recovery in Robot Programs," Languages for Sensor-Based Control in Robotics, U. Reubeld and K. Hoermeah, eds., Springer-Verlag, (1987).

7. Harmon, S. Y., "Dynamic Task Allocation and Execution Monitoring in Teams of Cooperating Humans and Robots," 1988 Workshop on Human-Machine Symbiotic Systems Proceedings, ORAU 89/C-140, CESAR-89/19, Oak Ridge, TN, (1988).

8. Parker, L. E., and C. R. Weisbin, 1988 Workshop on Human-Machine Symbiotic Systems Proceedings, ORAU 89/C-140, CESAR-89/19, Oak Ridge, TN, (1988).

9. Parker, L. E., and F. G. Pin, "Man-Robot Symbiosis: A Framework for Cooperative Intelligence and Control," Space Station Automation IV, W. C. Chiou, ed., Proc. SPIE 1006, 94-103, (1988).

10. Parker, L. E., and F. G. Pin, "Architecture for Dynamic Task Allocation in a Man-Robot Symbiotic System," Space Station Automation III, W. C. Chiou, ed., Proc. SPIE 851, 95-102, (1987).

11. Parker, L. E., and F. G. Pin, "A Methodology for Dynamic Task Allocation in a Man-Machine System," Methodologies for Intelligent Systems, Z. W. Ras and M. Zemankova, eds., North-Holland, New York, 488-495 (1987).

12. Parker, L. E., and F. G. Pin, "Dynamic Task Allocation for Man-Machine Symbiotic System," ORNL/TM-10397, June 1987.

13. Sacerdoti, E. D., A Structure for Plans and Behavior, North-Holland, New York (1977).

14. Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," Artificial Intelligence 16, 111-140 (1981).

15. Stefik, M., "Planning and Meta-Planning (MOLGEN: Part 2)," Artificial Intelligence 16, 141-170 (1981).

16. Vere, S. A., "Planning in Time: Windows and Durations for Activities and Goals," IEEE Trans. on Pattern Analysis and Machine Intelligence 5(3), 246-267 (1983).

17. Wilkins, D. E., "Domain-Independent Planning: Representation and Plan Generation," Artificial Intelligence 22, 269-301 (1984).

APPENDIX A

EXAMPLE ACTION OPERATOR RULES

```
Complete_Assembly(Benchmark)    PRECONDITIONS:
                    Positioned(Casing1,Jig_lower_center)
                    Positioned(Lever,Jig_axis)
                    Positioned(Spacer,Casing_1_bottom_edge)
                    Inserted(Large_pin_1,Casing_1_bottom_left_hole)
                    Inserted(Large_pin_2,Casing_1_bottom_right_hole)
                    Inserted(Large_pin_3,Casing_1_top_left_hole)
                    Inserted(Large_pin_4,Casing_1_top_right_hole)
                    Inserted(Peg,Casing_1_center_hole)
                    Positioned(Casing2,Jig_upper_center)
                    Inserted(Small_pin_1,Casing_1_bottom_side_left_hole)
                    Inserted(Small_pin_2,Casing_1_bottom_side_right_hole)
                    Inserted(Small_pin_3,Casing_1_top_side_left_hole)
                    Inserted(Small_pin_4,Casing_1_top_side_right_hole)
                    Inserted(Small_pin_5,Casing_2_bottom_side_left_hole)
                    Inserted(Small_pin_6,Casing_2_bottom_side_right_hole)
                    Inserted(Small_pin_7,Casing_2_top_side_left_hole)
                    Inserted(Small_pin_8,Casing_2_top_side_right_hole)
                    END
            DELETE_LIST:  Disassembled(Benchmark)
                          END
            ADD_LIST:  Assembled(Benchmark)
                       END

Complete_Disassembly(Benchmark) PRECONDITIONS:

                                    Positioned(Casing1,Casing1_home)
                                    Positioned(Lever,Lever_home)
                                    Positioned(Spacer,Spacer_home)
                                    Inserted(Large_pin_1,L_pin1_home)
                                    Inserted(Large_pin_2,L_pin2_home)
                                    Inserted(Large_pin_3,L_pin3_home)
                                    Inserted(Large_pin_4,L_pin4_home)
                                    Inserted(Peg,Peg_home)
                                    Positioned(Casing2,Casing2_home)
                                    Inserted(Small_pin_1,S_pin1_home)
                                    Inserted(Small_pin_2,S_pin2_home)
                                    Inserted(Small_pin_3,S_pin3_home)
                                    Inserted(Small_pin_4,S_pin4_home)
                                    Inserted(Small_pin_5,S_pin5_home)
                                    Inserted(Small_pin_6,S_pin6_home)
                                    Inserted(Small_pin_7,S_pin7_home)
                                    Inserted(Small_pin_8,S_pin8_home)
                                    END
                                DELETE_LIST:  Assembled(Benchmark)
                                              END
                                ADD_LIST:  Disassembled(Benchmark)
                                           END
```

```
Place(*object,*loc)          PRECONDITIONS:  Grasped(*object)
                                             Found(*loc)
                                             At(Hand,*loc>Hover_pos)
                                             END
                             DELETE_LIST:  END
                             ADD_LIST:  Positioned(*object,*loc)
                                        END

Grasp(*object)               PRECONDITIONS:  Handempty
                                             Found(*object)
                                             At(Hand,*object:Hover_pos)
                                             END
                             DELETE_LIST:  Handempty
                                           Inserted(*object,-)
                                           Positioned(*object,-)
                                           END
                             ADD_LIST:  Grasped(*Object)
                                        END

Insert(*object,*loc)         PRECONDITIONS:  Grasped(*object)
                                             Found(*loc>Hover_pos)
                                             At(Hand,*loc>Hover_pos)
                                             END
                             DELETE_LIST:  END
                             ADD_LIST:  Inserted(*object,*loc)
                                        END

Move_Arm(Curr_Loc,*to_loc)   PRECONDITIONS:  END
                             DELETE_LIST:  At(Hand,-)
                                           END
                             ADD_LIST:  At(Hand,*to_loc)
                             END

Release(*object)             PRECONDITIONS:  Grasped(*object)
                                             END
                             DELETE_LIST:  Grasped(*object)
                                           END
                             ADD_LIST:  Handempty
                                        END

Find(*object)                PRECONDITIONS:  END
                             DELETE_LIST:  END
                             ADD_LIST:  Found(*object)
                                        END
```

APPENDIX B

EXAMPLE JOB PLAN FOR CRANFIELD ASSEMBLY

Plan Number	Task Description
1	Find(Casing1)
2	Move_Arm(Curr_Loc,Casing1:Hover_pos)
3	Grasp(Casing1)
4	Find(Jig_lower_center)
5	Move_Arm(Curr_Loc,Jig_lower_center>Hover_pos)
6	Place(Casing1,Jig_lower_center)
7	Release(Casing1)
8	Find(Lever)
9	Move_Arm(Curr_Loc,Lever:Hover_pos)
10	Grasp(Lever)
11	Find(Jig_axis)
12	Move_Arm(Curr_Loc,Jig_axis>Hover_pos)
13	Place(Lever,Jig_axis)
14	Release(Lever)
15	Find(Spacer)
16	Move_Arm(Curr_Loc,Spacer:Hover_pos)
17	Grasp(Spacer)
18	Find(Casing_1_bottom_edge)
19	Move_Arm(Curr_Loc,Casing_1_bottom_edge>Hover_pos)
20	Place(Spacer,Casing_1_bottom_edge)
21	Release(Spacer)
22	Find(Large_pin_1)
23	Move_Arm(Curr_Loc,Large_pin_1:Hover_pos)
24	Grasp(Large_pin_1)
25	Find(Casing_1_bottom_left_hole>Hover_pos)
26	Move_Arm(Curr_Loc,Casing_1_bottom_left_hole>Hover_pos)
27	Place(Large_pin_1,Casing_1_bottom_left_hole)
28	Release(Large_pin_1)
29	Find(Large_pin_2)
30	Move_Arm(Curr_Loc,Large_pin_2:Hover_pos)
31	Grasp(Large_pin_2)
32	Find(Casing_1_bottom_right_hole>Hover_pos)
33	Move_Arm(Curr_Loc,Casing_1_bottom_right_hole>Hover_pos)
34	Insert(Large_pin_2,Casing_1_bottom_right_hole)
35	Release(Large_pin_2)
36	Find(Large_pin_3)
37	Move_Arm(Curr_Loc,Large_pin_3:Hover_pos)
38	Grasp(Large_pin_3)
39	Find(Casing_1_top_left_hole>Hover_pos)
40	Move_Arm(Curr_Loc,Casing_1_top_left_hole>Hover_pos)
41	Insert(Large_pin_3,Casing_1_top_left_hole)
42	Release(Large_pin_3)
43	Find(Large_pin_4)
44	Move_Arm(Curr_Loc,Large_pin_4:Hover_pos)
45	Grasp(Large_pin_4)
46	Find(Casing_1_top_right_hole>Hover_pos)
47	Move_Arm(Curr_Loc,Casing_1_top_right_hole>Hover_pos)
48	Insert(Large_pin_4,Casing_1_top_right_hole)
49	Release(Large_pin_4)
50	Find(Peg)
51	Move_Arm(Curr_Loc,Peg:Hover_pos)

Plan Number	Task Description
52	Grasp(Peg)
53	Find(Casing_1_center_hole>Hover_pos)
54	Move_Arm(Curr_Loc,Casing_1_center_hole>Hover_pos)
55	Insert(Peg,Casing_1_center_hole)
56	Release(Peg)
57	Find(Casing2)
58	Move_Arm(Curr_Loc,Casing2:Hover_pos)
59	Grasp(Casing2)
60	Find(Jig_upper_center)
61	Move_Arm(Curr_Loc,Jig_upper_center>Hover_pos)
62	Place(Casing2,Jig_upper_center)
63	Release(Casing2)
64	Find(Small_pin_1)
65	Move_Arm(Curr_Loc,Small_pin_1:Hover_pos)
66	Grasp(Small_pin_1)
67	Find(Casing_1_bottom_side_left_hole>Hover_pos)
68	Move_Arm(Curr_Loc,Casing_1_bottom_side_left_hole>Hover_pos)
69	Insert(Small_pin_1,Casing_1_bottom_side_left_hole)
70	Release(Small_pin_1)
71	Find(Small_pin_2)
72	Move_Arm(Curr_Loc,Small_pin_2:Hover_pos)
73	Grasp(Small_pin_2)
74	Find(Casing_1_bottom_side_right_hole>Hover_pos)
75	Move_Arm(Curr_Loc,Casing_1_bottom_side_right_hole>Hover_pos)
76	Insert(Small_pin_2,Casing_1_bottom_side_right_hole)
77	Release(Small_pin_2)
78	Find(Small_pin_3)
79	Move_Arm(Curr_Loc,Small_pin_3:Hover_pos)
80	Grasp(Small_pin_3)
81	Find(Casing_1_top_side_left_hole>Hover_pos)
82	Move_Arm(Curr_Loc,Casing_1_top_side_left_hole>Hover_pos)
83	Insert(Small_pin_3,Casing_1_top_side_left_hole)
84	Release(Small_pin_3)
85	Find(Small_pin_4)
86	Move_Arm(Curr_Loc,Small_pin_4:Hover_pos)
87	Grasp(Small_pin_4)
88	Find(Casing_1_top_side_right_hole>Hover_pos)
89	Move_Arm(Curr_Loc,Casing_1_top_side_right_hole>Hover_pos)
90	Insert(Small_pin_4,Casing_1_top_side_right_hole)
91	Release(Small_pin_4)
92	Find(Small_pin_5)
93	Move_Arm(Curr_Loc,Small_pin_5:Hover_pos)
94	Grasp(Small_pin_5)
95	Find(Casing_2_bottom_side_left_hole>Hover_pos)
96	Move_Arm(Curr_Loc,Casing_2_bottom_side_left_hole>Hover_pos)
97	Insert(Small_pin_5,Casing_2_bottom_side_left_hole)
98	Release(Small_pin_5)
99	Find(Small_pin_6)
100	Move_Arm(Curr_Loc,Small_pin_6:Hover_pos)
101	Grasp(Small_pin_6)
102	Find(Casing_2_bottom_side_right_hole>Hover_pos)

Plan Number	Task Description
103	Move_Arm(Curr_Loc,Casing_2_bottom_side_right_hole>Hover_pos)
104	Insert(Small_pin_6,Casing_2_bottom_side_right_hole)
105	Release(Small_pin_6)
106	Find(Small_pin_7)
107	Move_Arm(Curr_Loc,Small_pin_7:Hover_pos)
108	Grasp(Small_pin_7)
109	Find(Casing_2_top_side_left_hole>Hover_pos)
110	Move_Arm(Curr_Loc,Casing_2_top_side_left_hole>Hover_pos)
111	Insert(Small_pin_7,Casing_2_top_side_left_hole)
112	Release(Small_pin_7)
113	Find(Small_pin_8)
114	Move_Arm(Curr_Loc,Small_pin_8:Hover_pos)
115	Grasp(Small_pin_8)
116	Find(Casing_2_top_side_right_hole>Hover_pos)
117	Move_Arm(Curr_Loc,Casing_2_top_side_right_hole>Hover_pos)
118	Insert(Small_pin_8,Casing_2_top_side_right_hole)
119	Complete_Assembly(Benchmark)

APPENDIX C

EXAMPLE EXECUTION MONITORING TABLE

```
Subtask:  Find(Casing1)
          Preconditions:

          Continuing Conditions:


Subtask:  Move_Arm(Curr_Loc,Casing1:Hover_pos)
          Preconditions:

          Continuing Conditions:  Found(Casing1)


Subtask:  Grasp(Casing1)
          Preconditions:  Handempty
                          Found(Casing1)
                          At(Hand,Casing1:Hover_pos)

          Continuing Conditions:


Subtask:  Find(Jig_lower_center)
          Preconditions:

          Continuing Conditions:  Grasped(Casing1)


Subtask:  Move_Arm(Curr_Loc,Jig_lower_center>Hover_pos)
          Preconditions:

          Continuing Conditions:  Found(Jig_lower_center)
                                  Grasped(Casing1)


Subtask:  Place(Casing1,Jig_lower_center)
          Preconditions:  Grasped(Casing1)
                          Found(Jig_lower_center)
                          At(Hand,Jig_lower_center>Hover_pos)

          Continuing Conditions:


Subtask:  Release(Casing1)
          Preconditions:  Grasped(Casing1)

          Continuing Conditions:  Positioned(Casing1,Jig_lower_center)


Subtask:  Find(Lever)
          Preconditions:

          Continuing Conditions:  Handempty
                                  Positioned(Casing1,Jig_lower_center)
```

```
Subtask:  Move_Arm(Curr_Loc,Lever:Hover_pos)
          Preconditions:

          Continuing Conditions:  Found(Lever)
                                  Handempty
                                  Positioned(Casing1,Jig_lower_center)


Subtask:  Grasp(Lever)
          Preconditions:  Handempty
                          Found(Lever)
                          At(Hand,Lever:Hover_pos)

          Continuing Conditions:  Positioned(Casing1,Jig_lower_center)


Subtask:  Find(Jig_axis)
          Preconditions:

          Continuing Conditions:  Grasped(Lever)
                                  Positioned(Casing1,Jig_lower_center)


Subtask:  Move_Arm(Curr_loc,Jig_axis>Hover_pos)
          Preconditions:

          Continuing Conditions:  Found(Jig_axis)
                                  Grasped(Lever)
                                  Positioned(Casing1,Jig_lower_center)


Subtask:  Place(Lever,Jig_axis)
          Preconditions:  Grasped(Lever)
                          Found(Jig_axis)
                          At(Hand,Jig_axis>Hover_pos)

          Continuing Conditions:  Positioned(Casing1,Jig_lower_center)


Subtask:  Release(Lever)
          Preconditions:  Grasped(Lever)

          Continuing Conditions:  Positioned(Lever,Jig_axis)
                                  Positioned(Casing1,Jig_lower_center)1
```

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. G. A. Armstrong
3. S. M. Babcock
4. D. L. Barnett
5. M. Beckerman
6. P. F. R. Belmans
7. J. C. Culioli
8. G. de Saussure
9. F. Depiero
10. J. R. Einstein
11. C. W. Glover
12. W. R. Hamel
13. J. N. Herndon
14. J. P. Jones
15. H. E. Knee
16. F. C. Maienschein
17. W. W. Manges
18-22. R. C. Mann
23. J. R. Merriman
24. E. M. Oblow
25-29. F. G. Pin
30. D. B. Reister
31. B. A. Worley
32. EPMD Reports Office
33-34. Laboratory Records Department
35. Laboratory Records, ORNL-RC
36. Document Reference Section
37. Central Research Library
38. ORNL Patent Office

EXTERNAL DISTRIBUTION

39. Office of the Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations, P.O. Box 2001, Oak Ridge, TN 37831

40. John J. Dorning, Department of Nuclear Engineering and Engineering Physics, Thornton Hall, University of Virginia, Charlottesville, VA 22901

41. Robert M. Haralick, Department of Electrical Engineering, University of Washington, Seattle, WA 98195

42. Susan Hruska, Department of CSC, Florida State University, Tallahassee, FL 32306

43. James E. Leiss, 13013 Chestnut Oak Drive, Gaithersburg, MD 20878

44. Neville Moray, Dept. of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801

45-49. Lynne E. Parker, 125 Slade Street, Belmont, MA 02178

50. Carl Steidley, Central Washington University, Department of CSC, Ellensburg, WA 98926

51. Chuck Weisbin, MS/198-330, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109-8099

52. Mary F. Wheeler, Mathematics Department, University of Houston, 4800 Calhoun, Houston, TX 77204-3476

53-62. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831