PLANET: An Intelligent Decision Support System for Resource Planning in Manufacturing Organizations

Vasant Dhar
Graduate School of Business Administration
New York University

Harry E. Pople
Decision Systems Laboratory
University of Pittsburgh

May 1985

Center for Research on Information Systems
Computer Applications and Information Systems
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #92

GBA #85-24(CR)

Table of Contents

Abstract

This paper describes a problem solver called PLANET that has been developed in collaboration with a large computer manufacturing company to assist planning managers with the formulation and maintenance of planning models for resource allocation. PLANET is equipped with the primitives that enable it to *preserve* much of the richness of the *process* of the planning activity, namely, the generation of symbolic alternatives, and for the expression of domain specific knowledge which enables it to synthesize these alternatives into an overall planning model. This knowledge is maintained in a "meta-model." In contrast to modeling systems which allow for parametric perturbations of an algebraic model, PLANET's meta-model provides it with the capability for systematic variations in the symbolic model assumptions, with concomitant *structural* variations induced in the algebraic model that reflect the interdependencies of those assumptions. Whenever previously held assumptions change, PLANET uses the existing model as a point of departure in formulating the revised plan. In this way, the program is able to take cognizance of the *ongoing* nature of organizational problem solving, and can serve an important decision support function in maintaining and reasoning about evolving plans.

1 Introduction

The problem of long range planning is generic to most large business organizations. While the planning problem may involve aspects ranging from "strategic planning" to the more operational details of the firm, the fundamental problem is ultimately one of allocating an appropriate mix of resources (such as capital, labor, space etc.) in order to achieve desired goals and implement planned strategies. In organizations where the planning process is reasonably formalized, these goals and strategies may be reflected in a "descriptive plan". This forms the basis for the "resource plan" which is an indicator of the types and amounts of resources to be allocated over the horizon for which the plan is formulated.

In this paper, we describe the salient features of a problem solver called PLANET (Dhar, 1984) that has been designed in collaboration with a large computer manufacturing company (referred to here as CMC) to help planning managers with the formulation and maintenance of models for resource allocation. The investigation was initiated by CMC planning managers who expressed concern over the inadequacies of existing computer-based decision aids, and a serious need for an intelligent "planner's assistant" that might play an active role in supporting the resource planning activity in their manufacturing environment. This collaboration has yielded fruitful insights about some of the fundamental problems involved in resource planning, why traditional modeling systems prove to be of limited value in supporting this activity, and how these systems should be extended in order to overcome these limitations.

1.1 Fundamental Problems Involved In Resource Planning

Regardless of the context in which a resource plan is formulated, the fundamental problem of using such a plan for long range planning is that of maintaining reliability of the plan as an indicator of resources to be allocated, that is, one of ensuring that as events unfold, the plan continues to be an appropriate representation of reality over the relevant period of time. For various reasons, this often turns out to be a difficult problem. First, formulating plans involves making a series of projections which are invariably based on incomplete information about the future. These projections, which derive from the basic assumptions or premises on which the plan is based, are constantly subject to revision thereby calling for frequent adjustments to the projections that underly the plan. Further, in large organizations the process of formulating a plan typically involves several individuals from different functional areas or units of an organization. The "final plan" usually emerges after much discussion and negotiation among the various participants, requiring them to take cognizance of the interdependencies among the various parts of the plan. Any algebraic model derived from this plan to estimate specific resource requirements must be understood to be conditioned on the set of choices expressed in the plan. In effect, the model contains a particular view of the world in the domain being modeled -- that embodied by the set of relationships in the model.

In light of the inexact nature of the inputs that constitute the basis for the plan and the interdependencies among them, it is hardly surprising that a resource plan can tend to be extremely brittle; new information that violates some of its underlying assumptions can render even the most carefully crafted plan totally unreliable. While minor adjustments aimed at fine tuning a plan might sometimes constitute a workable solution, it is often the case that the new information calls for a significant restructuring of part or all of the original plan and its associated model. In order to be able to effect this change rationally, a manager must have explicit awareness of why prior decisions had been made, their underlying assumptions and rationale, their ramifications and interdependencies with various other facets of the model. In addition, he must have reasonable knowledge of cause and effect, time, space and resource constraints affecting his domain. Such supporting knowledge is essential if the decision maker is to gauge correctly what aspects of his conceptualization may have become unreliable based on the revised projections or new reality, and how to restructure the model in light of this new understanding.

1.2 The Role of Modeling Systems -- A Critique

Modeling systems[1] have been successfully used for various financial planning and other similar applications. One reason for this success is that they are to some extent non-procedural, making it easy for non-programmers to develop domain specific applications. Further, these systems allow for some fairly powerful kinds of parametric manipulations of an algebraic model, namely "sensitivity" and "target" analysis within a given model structure. However, they have some fundamental limitations as planning aids. Firstly, they lack the mechanisms necessary to direct the user's attention to relevant, focused aspects of a situation; instead, the responsibility of exploring the model lies entirely with the user. The consequences of the lack of an attention focusing mechanism are summarized by Reitman (1981) who contrasts the capabilities of current day modeling systems against those of a competent human decision support staff by posing the question "where do good alternatives come from?" Reitman notes:

[1] Sometimes referred to as "DSS Generators" because they provide primitives that can be used in order to build domain specific decision support systems (DSS).

> "A good human decision support staff has two jobs to do. First it must reduce the set of all possible actions to the few that look potentially realistic, feasible, and good. It is this small handful that the top level decision maker actually considers when he reaches his final decision. Second, both in winnowing through the alternatives, and in projecting their consequences, the staff somehow must deal directly with the interrelations among the various parties involved. This is the only way it can hope to apply its knowledge about the parties, their goals, their resources, and the constraints under which they must operate. In general, however, we simply do not yet know how to incorporate such knowledge in numerical projection models. As a result, there is a real ceiling to what we can expect of decision support systems cast in current molds." (Reitman, 1981).

Perhaps a more serious limitation of existing computer-based systems is their inability to take cognizance of the ongoing, evolutionary nature of organizational problem solving, that is, to preserve and reason about previous decisions and changes to them -- something that is an integral part of a manager's job. If we pose Reitman's question again, we realize that many good alternatives are in fact generated or synthesized in the course of formulating a plan. However, only a small subset of these become part of the "final" plan and reflected in the algebraic model that is derived from it. Unfortunately, much of the knowledge about issues and choices that were available, and the rationales for choosing or rejecting alternatives end up in filing cabinets or voluminous reports, often permanently. This is not altogether surprising. Given the effort involved in formulating the plan in the first place, and the difficulty of coordinating the diverse inputs from the various parties involved, a systematic assessment of the ramifications of changes can become overwhelming. Yet, in the absence of this knowledge, the existing algebraic model provided by a modeling system can have the effect of unnecessarily confining users to its limited view of an inherently flexible situation. For such problems, the real decision support needed is not in helping fine tune an existing model, but one of exposing a decision maker to the multiple perspectives brought about by changes in assumptions, and of interactively assisting in the reformulation model of the situation.

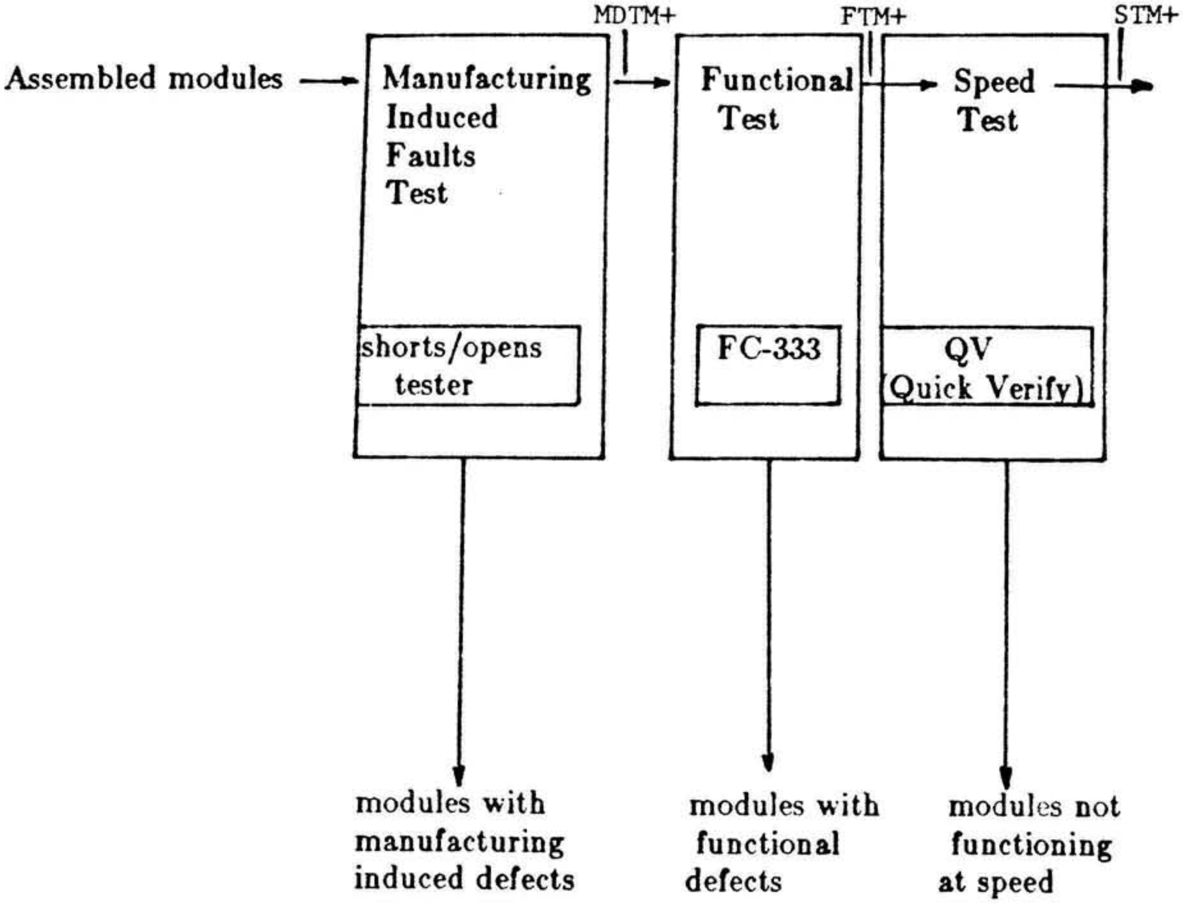
A major contribution of this research is that it provides a new methodology for resource planning. We have developed a problem solver called PLANET that can be used to create a planning model, which maintains an explicit awareness of the rationales for previous decisions, and their ramifications and interdependencies with other parts of the model. In contrast to the parametric perturbation facility provided by existing modeling systems, PLANET is equipped with a "meta-model" that provides it with the capability for systematic variations in the underlying model assumptions, with concomitant structural variations induced in the planning model that reflect the interdependencies of those assumptions. As a result, whenever previously held beliefs prove to be unjustified, the program has the potential to bring into focus new conceptualizations of the problem that account for the changed set of assumptions. A natural consequence of this approach is that the model underlying the resource plan continuously evolves over time -- PLANET always uses the "existing" model and the various justifications associated with it as a point of departure in formulating the "new" plan.

2 The Nature of Planning Assumptions

In the computer manufacturing industry, it takes several years from the time a product is planned to the point where a stable production process is in place. Planning for resource allocation through this time period requires making assumptions pertaining to many aspects of the business such as make-versus-buy decisions, where various components will be produced, the technology to be used, and what types of manufacturing processes will be employed to produce the products. Most of this information is recorded in the descriptive plan. Initially, there is considerable flexibility in the plan. As choices begin to firm up, they have the effect of constraining choices in related areas. Choices are also periodically revised in response to changing market conditions, internal policy issues, or resource constraints. In order to provide a flavor of the nature of assumptions/decisions involved, we shall now introduce some vocabulary pertaining to a small part of the CMC manufacturing process. This will also provide a context for the discussion in the remainder of this paper.

At CMC, the manufacturing process is typically partitioned into areas that deal with major components of the computer such as modules, kernels, subassemblies, cables and harnesses, peripherals, and various customer-specific options. The manufacturing plan contains information on all such major areas, each of which involves performing various functions. For example, testing a module requires various types of diagnostic procedures. First, the assembled module could be tested for faults that may have been induced as a result of assembling the module components. Typically, these consist of shorts and open circuits. We refer to this part of module testing as the Manufacturing Induced Faults Test (MIF-Test). Further testing may also be necessary to ensure that the components of the module are functional, i.e. perform within tolerance limits. This is called the Functional test (also referred to as a functional in-circuit test). Finally, the complete module may be tested to confirm that it performs satisfactorily under conditions it will have to endure in the finished product. This process, sometimes referred to as "testing modules at speed," is labeled Speed test. Thus, the overall module test function could involve the configuration[2] of activities shown in Figure 1. Entries in the lower boxes are methods to be used in order to perform each of the diagnostic activities. Taking into account other functions, a visual representation of the descriptive plan would begin to resemble that shown in Figure 2.

[2] A configuration may be viewed as a typical arrangement of activities, much in the way that a molecule consists of a typical arrangement of atoms.


MDTM+
FTM+
STM+
Assembled modules
Manufacturing Induced Faults Test
shorts/opens tester
Functional Test
FC-333
Speed Test
QV (Quick Verify)
modules with manufacturing induced defects
modules with functional defects
modules not functioning at speed

Legend:

MDTM+: Modules passing MIF test
FTM+ : Modules passing functional test
STM+ : Modules passing speed test

Figure 1
(An assumption -- a schematic of a configuration of activities and methods of testing to be used in module testing -- taken from the CMC Manufacturing plan. This schematic represents one of several ways that the function can be organized. Similarly, for each of the (diagnostic) activities above, several methods of testing are typically available.)

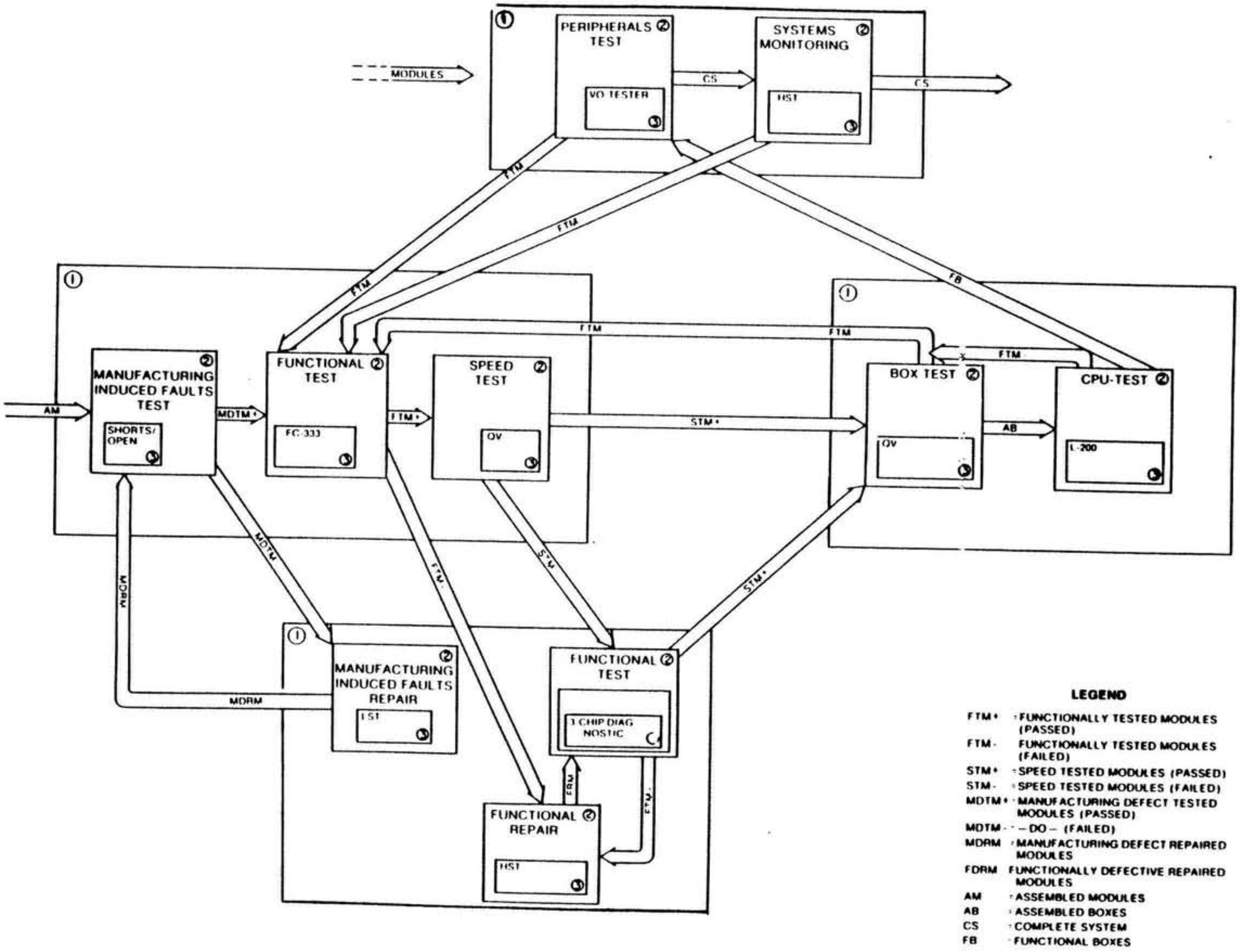


FIGURE 2. (INITIAL CONFIGURATION)

(Schematic of part of a manufacturing plan)

In addition to broad assumptions about configurations that can be used, the plan also consists of more detailed assumptions about the manufacturing process. For example, once a certain configuration -- such as the one illustrated above -- is assumed within an function, lower level assumptions follow:

* The QV diagnostics software to be acquired from Hitech Corporation will be able to test modules at speed. If not, this will have negative impacts on test and repair times, capital equipment, and manpower estimates.

* The QV diagnostics software will achieve fault isolation to the 3 chip level within the module.[3]

It is generally the case that the more abstract assumptions within a plan change less often than the details. For example, a decision to go with a high volume configuration of activities (a policy oriented or market driven assumption) is likely to be revised less often than an assumption about what tester to use for an activity or how to use it.

Once the set of assumptions that constitute the descriptive plan are in place, an algebraic model can be formulated and represented using a modeling system, and resource requirements derived by solving the set of algebraic relationships that correspond to the chosen set of assumptions.

[3] Diagnostics software basically speeds up the module diagnosis process, i.e. the process of localizing faults in a failing module to a specific part of the module, for example, to within 3 chips. In the absence of such software, although 3 chip isolation may still be possible to achieve, it is likely to require other types of resources such as skilled labor and higher test times per module. This reallocation of resources may also affect choices in other parts of the plan.

2.1 The Meta Model -- I

Conceptually, the plan in Figure 2 can be viewed as being a specialization of a more general model that incorporates knowledge about the range of alternatives known to the various decision makers within the organization. This meta-model, corresponding to plans like the one shown in Figure 2, can be conceptualized as a hierarchy of alternatives as shown in Figure 3. In addition to knowledge about alternatives, the meta-model also contains "action oriented" knowledge that enables it to synthesize these alternatives in various ways. This latter type of knowledge is described in section 4.

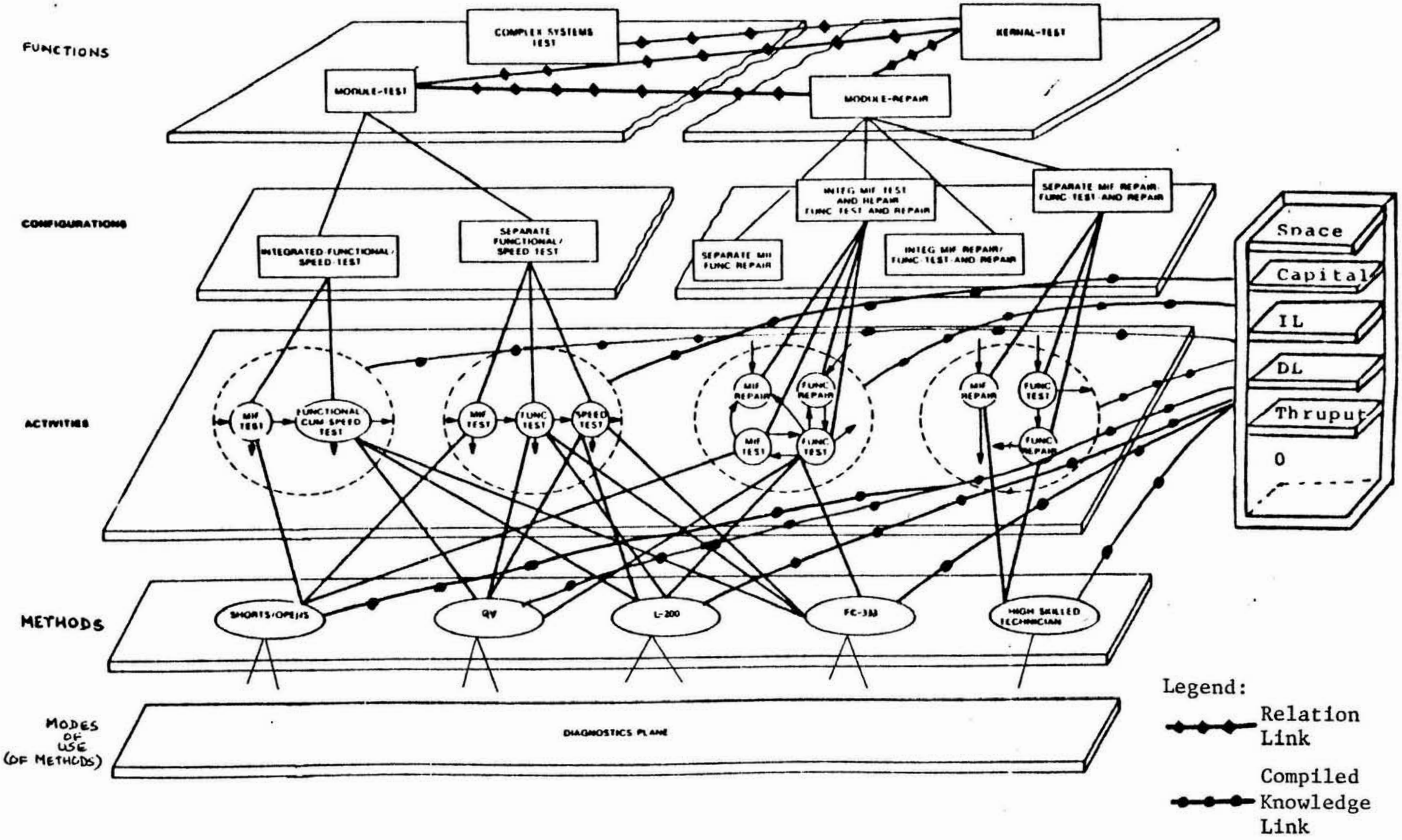
Like module-test, other functions can also be organized in terms of different clusters of activities which can employ different methods. Each of these sets of choices is represented as a plane in the overall hierarchy.[4] The column of boxes on the right represents an "objectives vector" which will be explained in the following sections.

The basic idea of organizing knowledge hierarchically in terms of a meta-model that "knows about its own knowledge" is not new in the AI literature. Basically, the meta-model can be viewed as a

> "compact, high level description of structure, organization, or content that may be used both to provide a framework for lower level processing, and to express expectations about the world." (Davis, 1979, p.419.)

In effect, a meta-model should provide a program with an "introspective" ability, that is, the ability to reason about its lower level knowledge. Viewed in this way, the meta-model may be regarded as a generalization of a lower, "object level" model where individual differences among the lower level models emerge as a result of

[4] In Figure 2, the (1) corresponds to the configuration plane, (2) to the activity plane, (3) to the method plane, and (4) to the mode of use


FUNCTIONS
COMPLEX SYSTEMS TEST
KERNAL-TEST
MODULE-TEST
MODULE-REPAIR
CONFIGURATIONS
INTEGRATED FUNCTIONAL/ SPEED TEST
SEPARATE FUNCTIONAL/ SPEED TEST
INTEG MIF TEST AND REPAIR FUNC TEST AND REPAIR
SEPARATE MIF REPAIR FUNC TEST AND REPAIR
SEPARATE MIF FUNC REPAIR
INTEG MIF REPAIR/ FUNC TEST AND REPAIR
ACTIVITIES
MIF TEST
FUNCTIONAL CUM SPEED TEST
MIF TEST
FUNC TEST
SPEED TEST
MIF REPAIR
FUNC REPAIR
MIF TEST
FUNC TEST
MIF REPAIR
FUNC TEST
FUNC REPAIR
Space
Capital
IL
DL
Thruput
0
METHODS
SHORTS/OPENS
QV
L-200
FC-333
HIGH SKILLED TECHNICIAN
MODES OF USE (OF METHODS)
DIAGNOSTICS PLANE
Legend:
Relation Link
Compiled Knowledge Link

varying certain facets of the meta-model. In a somewhat different mode, the meta-model can also be used for recognizing and classifying new object level instances in accordance with its general precepts about the domain. This requires it to maintain a significant amount of knowledge about the structure and purpose of the lower level model. PLANET's meta-model is used in both these forms. However, due to space limitations, we shall restrict ourselves to the first type of situation in this paper; issues having to do with computer assisted knowledge acquisition are dealt with in Dhar (1984).

3 System Behaviour

In this section, we shall provide a flavor of PLANET's power as a decision support tool by presenting annotated excerpts of system behavior. Since there is usually an existing plan in place,[5] a dialogue typically begins with the retraction of one or more assumptions that are part of the fully formulated plan. A reassessment of an existing plan may also become necessary if additional knowledge is added to the program possibly as a result of changes in the underlying set of alternatives in the various parts of the manufacturing process. In such circumstances, because the meta-model itself changes, it is likely that the manufacturing plan will also require adjustments.

It is important to stress a few points about how the responsibility for generating a new plan can be distributed between the user and the program in the man-machine interaction. At one extreme, we could have a scenario where the program assumes the initiative with little guidance from the user. On the other hand, it is often the case that a user has in mind a general outline of the plan that is to be formulated,

[5] The program can also generate a plan "from scratch" if one does not already exist.

along with a possible sprinkling of low-level details (for example, "use an L-200 tester in the Module-repair process at all costs"). Under these circumstances, the program must "do its best" while taking cognizance of the constraints specified by the user. In effect, the program's responsibility becomes one of instantiating the appropriate low-level aspects of the plan within the specified framework. The constraints can be specified at any point in the dialogue -- at the beginning if the user is reasonably certain about what the constraints are to be, or at various points during the dialogue as the various facets of the problem begin to move into focus. Regardless of the specific differences in the various forms of interaction possible, it is important that the program be flexible enough to provide the user with the potential to direct the program's behavior to whatever extent and level of detail desired.

In the following dialogue, we shall use part of a typical plan (shown in Figure 2) as a point of departure, violating an assumption that had played a major role in the formulation of that plan. That plan was based on several assumptions, a critical one being that there would be micro diagnostics (diagnosing failing modules to the 3 chip level using a special tester) available in the module repair process. Having micro diagnostics usually reduces the volume throughput of a process because detailed diagnostics are more time consuming than macro level diagnostics. In order to avoid redundant test processes, this assumption had in turn led to the choice of macro diagnostics (board level diagnostics) in the module test making it a high volume process.[6] Since the micro diagnostics assumption played a significant role

[6] Functionally failed modules from other parts of the manufacturing process were therefore returned to the module test area (components are typically sent to high volume before they are directed to other, lower volume processes that can also handle them).

in the formulation of the initial plan, let us examine the consequences of withdrawing this on the overall plan. The example has been simplified by considering a limited section of the task environment. Complicating the problem with domain-specific details is likely to contribute little toward illustrating the basic characteristics of the problem solver.

In order to demonstrate clearly the progressive constraint posting that characterizes the plan formulation process, we make use of what is commonly referred to in Artificial Intelligence as a state-space. The topmost box in Figure 4 represents an initial "state" that corresponds to an existing plan, in this case, the one shown in Figure 2. This state is the end result of a previously generated state-space that resulted in the existing plan. When one or more assumptions corresponding to this plan are violated, some parts of the plan become "undone" and must be reconfigured.[7] The "affected areas" are shown in the second state from the top in Figure 4. The goal is to "reconfigure" the affected areas as effectively as possible into a new plan that takes cognizance of the changes in assumptions. Figure 4 provides a graphical representation of the evolving state-space. Internally, a state is a structured object data structure that contains partial formulations of the various parts of the plan being restructured.

The following excerpt of the user/system dialogue resulted from the violation of the micro diagnostics assumption. Asterisks represent system prompts where the user has the opportunity to enter a command or data. Regular type is system output and the text within parentheses explains the dialogue. User input is in bold capitals.

[7] This requires the program to maintain "dependency information" (Stallman and Sussman, 1977) from the previous state-space.

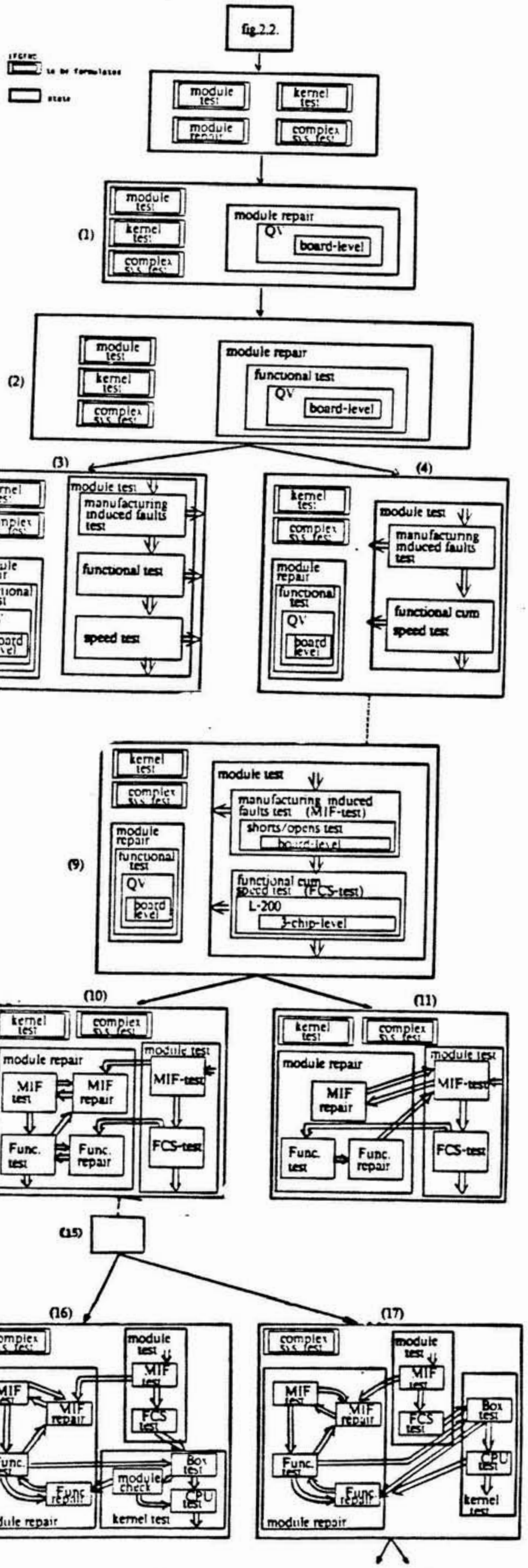


Figure 4 (State Space Search)

(The states illustrate a synthesis of choices being made in the various areas into an overall manufacturing plan.)

```
* EXAMINE MODULE-REPAIR
```

(The examine command allows the user to look at the ''existing plan'' i.e. the configuration in place, methods chosen and how they are being used, and the flows of material (inputs and outputs) among the various sections of the plan. Here, the user is examining the plan shown in Figure 2.)

```
configuration:  integrated-MIF-repair/functional-test-and-repair
more?  YES
For activity: MIF-repair...
INPUTS:
  1:
   input: manuf-defect-modules
   from: (module-test MIF-test)
   units: 110.0
OUTPUTS:
  1:
   output: manuf-repaired-modules
   amount: 110.0
current method candidates: nil
methods eliminated from consideration: nil
method chosen: low-skilled-technician

For activity: Functional-repair...
```

(These ''input/output'' relationships correspond to the overall configuration shown in Figure 2. The number of units flowing through each of the activities was calculated using yield estimates of the various types of modules in those activities. The ''method candidates'' slot is nil since a method choice for each of the activities has already been made. The ''methods eliminated from consideration'' is more interesting to examine during plan reformulation where it shows a list of foreclosed choices and reasons for why these were eliminated.)

```
* ACCEPT
```

(With this command, the user can enter changes in assumptions and/or specify certain constraints within which the program is to formulate a plan.)

```
**  NOT (MODULE-REPAIR 3-CHIP-DIAGNOSTICS)
**  CONSTRAIN (MODULE-REPAIR QV BOARD-LEVEL-DIAGNOSTICS)
**  DONE
```

(Here, we simply violate an existing assumption. In addition, a constraint was imposed, that there be macro diagnostics (board level) using the 'QV' type of tester in module-repair. In the absence of such a constraint, the program could at a later point, choose not to have any kind of diagnostics in module-repair at all. The ''NOT'' and ''CONSTRAIN'' are part of a controlled vocabulary used in the user/system interaction. The program is also equipped to respond to more general types of constraints that may of more interest at higher managerial levels such as ''OFFLOAD module-test'' in which case it works with a modified ''top level'' situation.)

As a result of the changes the following are affected:

- Module-test
- Module-repair
- kernel-test
- complex-systems-test

(Violating the micro diagnostics design choice had the effect of ''loosening up'' other parts of the plan (these areas are enclosed in the double line boxes in Figure 4). For example, an alternative, lower throughput configuration than the existing one was considered and chosen in module-test. This in turn led to the reconsideration of an alternative, higher volume configuration in module-repair that had been ruled out in the earlier plan because of the high-volume configuration in module-test (it is not usually prudent (or necessary) to have high volume processes in module-test and module-repair). Other parts of the plan also required appropriate modification. Let us illustrate some of these changes by looking at some parts of the interaction. State-1 in figure 4 corresponds to the constraint specified by the user.)

```
Select: state-2
                function: module-repair
                activity: Functional-test
                method: QV
                mode-of-use: board-level
```

(Specifying the need for diagnostics using the 'QV' tester establishes 'functional-test' as one of activities that will be part of the module-repair function. In the states shown in Figure 4, nested boxes (except those corresponding to the 'still to be formulated' functions) correspond to the various levels of the meta-model of Figure 4. State identifiers (i.e. state-2) make it easy to refer to particular states in the state space, especially for purposes of explanation.)

```
* WHY-CHOOSE STATE-2
(module-repair Functional-test QV board-level) because
Functional-test is the only activity in module-repair that is
relevant for the board level of diagnosis.
```

(The program infers that since a decision was made to have some kind of diagnostics take place in module-repair, it is necessary to have 'functional testing' since that is the only diagnostics activity that can currently occur in module-repair (others could have existed at some point in time). Note that the '*' prompt above indicates that the program has reached a ''quiescent state'' which means that it is unable to make any more inferences (or ''propagate further constraints'') based on the facts it has thus far. In figure 4, this corresponds to a branching/choice point in the state space).

```
More? YES
(module-repair board-level) because
YOU told me so.
```

(These explanations were generated using PLANET's 'dependency network' which we explain in the next section. Basically, this involves keeping track of justifications for conclusions, which in turn can serve as justifications for further decisions made by the program. English like explanations are generated using various ''templates'' that correspond to different types of inference methods that the program is familiar with. In the example above, the program used a ''premise'' template that corresponds to user-specified constraints, and a special case of a ''all other alternatives eliminated'' template. Similarly, as we shall show, there are other types of templates corresponding to situations where the conclusion may be the result of applying a rule, or where the program chooses among several competing alternatives.)

```
* GO

Option:
        function: module-test
        configuration: integrated-functional/speed-test

Option:
        function: module-test
        configuration: separate-functional/speed-test
```

(These are the options shown as states 3 and 4 in the state space.)

```
* SHOW-CHOICE
Chosen Option:
        function: module-test
        configuration: integrated-functional/speed-test
```

(The process of how this choice was made will be explained momentarily. Let us continue the dialogue at state-5 (not shown in figure 4) where the program begins to make some ''lower level'' choices within the integrated-functional/speed-test configuration of the module-test function.)

```
o
o

MIF-test has only one tool candidate left.
Therefore,
Select: state-5
        function: module-test
        activity: MIF-test
        tool: shorts/opens
Select: state-6
        function: module-test
        activity: MIF-test
        tool: shorts/opens
        level-of-diagnosis: board-level
Select: state-7
        function: module-test
        activity: Functional-cum-speed-test
        tool: L-200
```

(The behavior above (not shown in figure 4) illustrates a succession of choices of the program operating in a 'constrained mode', i.e. where it can make choices about configurations, activities, tools etc. based on the information it has. To the extent possible, the program attempts to operate in this mode. However, as we have shown, it periodically runs into situations where it is unable to make further inferences based on the facts it has.)

```
* GO

Option:
       function: module-test
       activity: Functional-cum-speed-test
       tool: L-200
       level-of-diagnosis: 3-chip-level
Option:
       function: module-test
       activity: Functional-cum-speed-test
       tool: L-200
       level-of-diagnosis: board-level
* PURSUE 1
```

The '1' refers to the first option above. In effect, the user instructs the program to pursue a certain alternative regardless of the program's choice and reasons for it. The alternative chosen by the user is shown as part of state-9 in the state space. For the sake of visual clarity, we will omit the lower level details in the state following this one, assuming that these details are ''carried over'' from previous states. Let us pick up the interaction at state-9 in Figure 4 where the program has once again reached a quiescent state and must therefore enumerate its choices in some part of the plan,[1] and choose among them.)

```
* GO
Option:
     function: module-repair
     configuration: integrated-MIF-test-and-repair/functional-test-and-repair
Option:
     function: module-repair
     configuration: separate-MIF-repair/functional-test-and-repair
* SHOW-CHOICE
Chosen Option:
     function: module-repair
     configuration: integrated-MIF-test-and-repair/functional-test-and-repair
* WHY-CHOOSE
In comparison to:(module-repair separate-MIF-repair/functional-test-and-repair)
Advantages in terms of:
       1: decreasing WIP
       2: increasing throughput
Disadvantages in terms of:
       1: increasing capital
In addition, the following were eliminated:
--separate-MIF-repair/functional-repair because:
  the activity Functional-test, is not part of this configuration.
--integrated-MIF-test-and-repair/functional-test because:
  low-volume configuration in module-test eliminates
  low-volume configurations in module-repair.
```

[1] PLANET's decision about what part of the manufacturing process to focus on (to make a forced choice) takes place on the basis of a ''precedence ordering'' that reflects the importance of the various areas of the manufacturing process.

(When forced into making a choice where several alternatives exist, i.e. a choice/branching point in the state space, the program uses a heuristic evaluation function that compares the remaining alternatives with respect to that decision (in the case above, two alternatives) across a ''vector of objectives'' (shown on the extreme right in Figure 4). The vector is composed of elements such as capital required, throughput, space required, direct labor required etc. Alternatives are ranked according to the extent to which they contribute to these elements. PLANET uses target values for some of these elements. For example, it has a space goal that corresponds to the amount of space that is currently available at CMC -- which cannot be exceeded. Similarly labor has a goal number although deviations on either side of it are undesirable. Capital does not have a goal figure -- alternatives requiring less capital are preferred, ceteras paribus, to more capital intensive ones. Similarly, in situations where there is not excess capacity in place and where throughput is a factor, higher throughput configurations would be favored over low throughput ones. The ultimate choice is based on which alternative emerges as the most 'balanced' across the various attributes.

As we can also see in the situation above, because of the first constraint imposed by the user, namely that of having board level diagnostics in module-repair, functional testing became necessary there, which in turn ruled out 'separate-MIF-repair/functional-repair' because of the lack of a testing activity in that configuration. The second elimination above was the result of an application of what we refer to as a ''heuristic rule'' which ruled out all low volume configurations in module-repair (in this case the one mentioned above) because a low volume configuration (integrated-functional/speed-test) had been established in module-test -- the two functions are 'complementary' in that at least one of them must usually be geared toward diagnosing failed modules in high volumes. Again, standard templates were used in order to generate these two explanations.)

```
o
o
Couldn't find a potential acceptor in module-repair
for manuf-defect-modules coming in from module-test.
Do you want to:
        1) Specify this information now, or
        2) Let me formulate the plan without this information.
        ** 1
        Okay -- specify which activities are potential acceptors in
module-repair for manuf-defect-modules: MIF-TEST
```

(This message points out a lack of 'input/output' knowledge in the program's knowledge base. Messages such as this were quite common during PLANET's early stages and were extremely useful in helping us identify missing sections in its knowledge base. As we see, the program is equipped with knowledge about its data structures, and mechanisms that allow it to detect these, and to allow the user to interactively 'fill in the gaps.' It should also be apparent by now that the program is doing a fair amount ''behind the scenes'' such as attempting to set up the various input/output relationships (of materials) among activities as the different parts of the plan become specified. As we showed in the beginning of the dialogue, the user can examine (and change) these relationships during the course of the interaction.)

The plan that was finally formulated appears in Figure 5. Broadly speaking, the configurations chosen in module-test and module-repair changed with some differences in lower level details and input/output relationships becoming necessary as well. This changed model was then used to generate a structurally modified resource plan reflecting the changed resource requirements and reasons for these changes. In summary, new developments pertaining to the level of diagnostics to be used in a certain part of the manufacturing process -- a type of situation that occurs periodically -- caused a significant amount of restructuring of the original model. Let us now briefly describe some of the data structures and mechanisms that result in the type of behavior just described.

4. The Meta-Model -- II

In section 2, we provided a graphical description of part of the meta-model that contains the range of alternatives known to decision makers involved in the planning process. We now provide a flavor of the data structures and the "action oriented" features of the meta-model that are instrumental in threading together the various assumptions and choices into an overall manufacturing plan.

The knowledge of the meta-model resides in a "society of agents" designed to represent known standard areas of the planning activity or individual human specialists in the different functional areas of the organization who have responsibility in the planning process. In the limited part of the manufacturing environment that we have included in this discussion, the relevant specialists are organized hierarchically as shown in Figure 3. "High-level" specialists correspond to functions such as module-test and kernel test. These are in turn composed of lower level specialists whose knowledge corresponds to the more detailed aspects of the plan. Each of the different types of specialists, such as the "activity" type or the "configuration" type, corresponds to a plane in Figure 3.

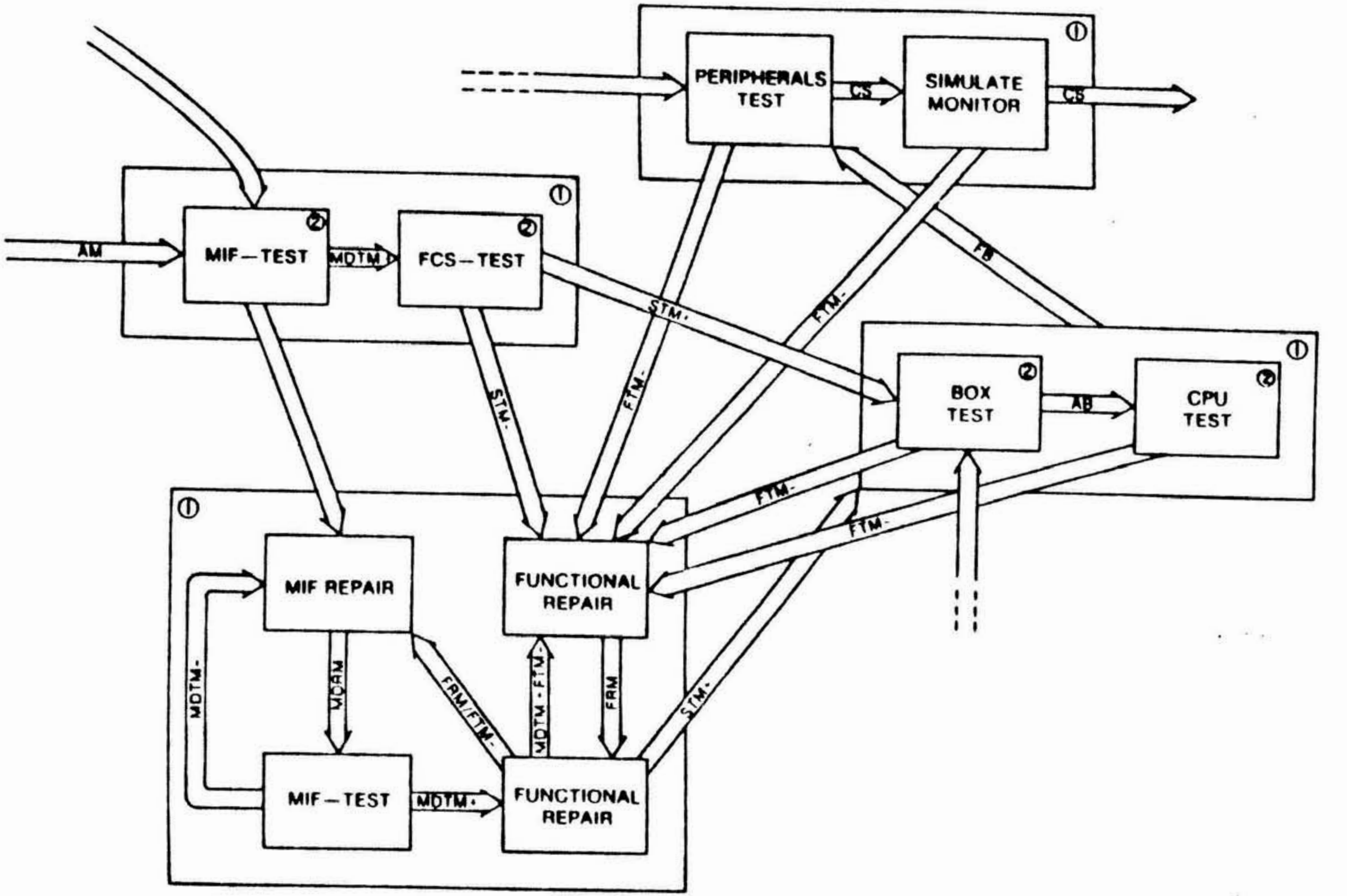


Figure 5
(A Schematic of part of the reformulated Manufacturing plan after the assumption of micro diagnostics in module test is withdrawn. In this figure, only "configuration" changes in each of the functions and some changes in the "input/output" relationships of material among the various activities are shown. Lower level changes (in methods of testing/repairing, and the modes of use (such as 3-chip level diagnostics etc.) for the various methods are not shown here in order to keep the figure readable.)

PLANET's specialists are represented as "objects" in HOUSE (Quayle, 1982), a Franz Lisp object oriented programming environment that is similar in spirit to the FLAVORS package (Moon and Weinreb, 1981). Basically, an object oriented program models a problem in terms of data structures called objects that correspond to the real world entities in the domain under consideration. Responsibilities of an object (which corresponds to a domain specialist) include responding to decisions being taken in other parts of the manufacturing environment and communicating its decisions so that other specialists may also make appropriate adjustments to their parts of the plan (see Hewitt, 1976 for a general discussion of the "message passing" protocol in object oriented programs). These "adjustments" are carried out using "heuristic rules" and "intra-specialist procedures," both to be discussed shortly. Other, book-keeping oriented responsibilities of a specialist include keeping track of its current choice (with respect to whatever decision(s) for which it is responsible), reasons for it, and possible alternatives to the existing choice.

4.1. The 'Compiled-knowledge'

The process of plan formulation, as we have shown, consists of a sequence of moves through a state-space with several alternatives potentially available at various states. In assessing the potential promise of a choice (state) before the details are known, PLANET performs a heuristic evaluation[9] of the various states by using "macro-level" or "compiled" knowledge about the domain. Basically, this knowledge consists of high-level associations (indicated by the solid dotted lines in Figure 3) that compare the various alternatives on how they compare on the existing

[9] The problem of having to evaluate "potential states" at early stages of a state-space is a classic problem in Artificial Intelligence. See Nilsson (1981) for an excellent discussion on the A* and other algorithms that address this problem.

resource scenario of the organization. Let us consider a few examples of some of these associations which we have extracted from CMC planning documents:

a) "Functional/speed testing in module testing usually involves a higher screen time per module than using separate functional and speed tests."

> (The screen time estimate can then be used to figure out, at least qualitatively, how alternate configurations would do with respect to "throughput" among the various parts of the manufacturing process. The estimate is heuristic since exact screen time estimates will only be known once the low level details of the plan have been determined.)

b) "Increasing WIP (work in process) by one week costs over a half percentage point in the internal rate of return at constant volume of product shipped."

> (This example indicates the sensitivity of a financial measure (IRR) to changes in some of the controllable aspects of the manufacturing process assuming a certain model of the firm's external environment. Because PLANET's knowledge base incorporates knowledge about the manufacturing plan at various levels of abstraction, it becomes possible for managers to assess possible consequences at the manufacturing level of varying (or attempting to achieve) the financially oriented goals of the organization.)

Although these types of evaluations are not usually very precise, especially when the details of a plan are yet to be determined, they serve a critical function in focusing the program's attention on those areas of the state-space that appear intuitively to be the most promising. This macro-level, or compiled knowledge is indicated by the dotted links in Figure 3. Each link represents one or more of the types of statements outlined above.

4.2. The 'Relation' Link and 'Heuristic Rules'

Whenever a PLANET specialist engineers a change in its area of responsibility, it informs other specialists[10] that are likely to be affected as a consequence of the

[10] This inter-specialist communication is implemented using a FLAVORS type message passing protocol.

change. The relation link is used for this purpose in order to dispatch a message about the change to the relevant specialists. An "affected" specialist reacts to messages by using its local expertise, some of which is represented in the form of "heuristic rules" that are sensitive to changes in problem context. For example, a heuristic rule used by the specialist associated with module-testing is:

"if micro level diagnostics have been established in module-repair, rule out micro-level diagnostics in the module-test process."

All heuristic rules of this type have some sort of a basis which represents a general rationale for the existence of the rule. In the rule above, the basic idea is to avoid replication of expensive and time consuming diagnostic procedures in the two functions.[11] Similarly, different functions may be related according to other bases, an obvious one being 'volume'--it would be unwise to choose a "low volume configuration" for a function if processes preceding it are likely to be set up for high throughput. In effect, a rule does not make reference to individual alternatives (such as "L-200"), but to more general criteria which determine whether individual alternatives such as L-200, 3-chip diagnostics, board level diagnostics etc. are to be chosen or eliminated from further consideration. This is important because we would not like to modify the rules whenever new alternatives are added or removed from the knowledge base. Instead, we classify the new alternative according to the relevant bases by specifying some fairly general characteristics about it, and leave the rules unaffected.

In formulating a plan, much of a specialist's activity is oriented toward

[11] Likewise, a decision to have macro diagnostics in module repair rules out macro diagnostics in module test and vice-versa -- at least one of them usually has micro diagnostics.

successive refinement of its set of options. Thus, the action part of a heuristic rule is an elimination of a class of alternatives (which have been identified according to some basis) rather than an explicit selection of one.[12] Some elimination of alternatives also takes place via constraint propagation in the alternatives hierarchy. For example, as we showed in the dialogue, certain configurations not containing an activity specified by the user were eliminated as potential candidates in module-repair. Similarly, constraints can also be propagated downward in the hierarchy. A natural consequence of this reasoning by elimination process is that a specialist must be equipped with the machinery that enables it to propagate constraints in the relevant parts of the alternatives hierarchy, and to realize when only one alternative remains. This is accomplished via "intra-specialist procedures."

4.3. Explanation and Dependency Directed Reasoning

It is widely recognized that the credibility of a problem solver is heavily dependent on its ability to explain its actions to a user. An important theme running through much of AI research in the last decade is that of "self aware" problem solvers that are able to "account for their actions." (Stallman and Sussman, 1977). The basic idea is that a "rational" problem solver must maintain "dependency information" to justify its beliefs with acceptable reasons.

Dependency information serves two related functions. First, it can be used by a program internally, to examine and revise its set of beliefs as is the case when previously held assumptions are no longer valid. When only a few previous choices

[12] The basic rationale for reasoning by elimination is that the "confirming evidence" required by this method of reasoning is usually less precise than that required in order to make an unequivocal choice. We have observed that managers often find it much easier to specify alternatives that are unacceptable under various circumstances than to make a specific choice.

in a large model change, dependency information can be used to undo and reformulate only the affected parts of the model. A few such dependencies are shown in Figure 4.[13]

Dependencies can also be used externally, to provide a user with explanations or rationales for existing choices. Explanations can play a critical role in the development and evaluation of a program's knowledge base. If the knowledge base is to grow over time, it is important to be able to assess the veracity of the program's reasoning; the only practical solution to the problem, especially as a program begins to use large amounts of diverse types of knowledge in its reasoning process, is for it to be able to explain itself.

In addition to being used for purposes of system evaluation and development, explanations are essential in order to convince a user of the rationality of an action. In some contexts, notably Computer Aided Instruction (CAI), a naive user is likely to find explanations educational; at the other extreme, an expert user in some domain will probably use the explanation as a reassurance of the program's actions or in detecting discrepancies in the program's line of reasoning (Davis, 1982). In the type of planning situation considered here, we could expect explanation to play both these types of roles. Since PLANET's knowledge does not correspond to any single expert but has been coalesced from different individuals in the organization, it can be expected that there will be numerous situations where the program will use knowledge pertaining to areas where a user may not be an

[13] The choice in state-2 had eliminated certain configuration candidates in module-test. When a configuration for module-test was finally chosen using a heuristic evaluation function (state-10), the choice turns out to be dependent on the choice corresponding to state-2 because otherwise, one of the eliminated configurations might have been preferred to the chosen one.

expert. Under these circumstances, explanations are more likely to be informative rather than simply reassuring.

PLANET's explanations are of three types. First, some of its inferences are the result of rule applications. Here, the justification for a conclusion is essentially the instantiated left hand side of the rule. This type of explanation facility is central to languages such as AMORD (de Kleer et.al, 1977). In addition to maintaining such dependencies that result from rule applications, PLANET also keeps track of eliminations and conclusions that result from the intra-specialist procedures. Finally, in the heuristic mode, where its basis for making a choice is the weakest, the "explanation" for a choice includes a summary of tradeoffs among the various alternatives. In the dialogue, we showed a situation where all the three types of justifications are part of the total explanation provided by the program. First, a summary of tradeoffs was listed since the program was in the heuristic mode. This was followed by a list of eliminations, the first being the result of a rule application, and the second one an elimination via an intra-specialist procedure. These basic types of explanations, which stem from PLANET's problem solving strategies, provide a reasonably rich vocabulary for explanation.

5. Conclusion

The PLANET framework represents a step toward a genre of decision support systems that can play an active role in supporting complex decision making in the managerial context. Let us briefly summarize the reasons for its power as a decision support tool.

PLANET gains much of its power from its ability to collect, preserve and manipulate a store of domain-specific knowledge encapsulated in the form of the meta-model. An important feature of the meta-model is that its knowledge is represented at various levels of abstraction, allowing a user to engage in a dialogue at the level of detail desired. This feature can have far reaching implications for decision making at all levels of an organization. On the one hand, the program can be used by managers at the operational level in dealing with lower-level engineering and manufacturing constraints. At the other extreme, it can also be used by high level planning managers to analyze alternatives that are likely to have an impact on the strategic operations of the organization. As we have observed in the dialogues, PLANET takes great care in preserving whatever alternate options it generates at various points in the planning process along with the relative advantages and disadvantages of these alternatives measured against a vector of objectives. Finally, its flexible interface makes it easy for the user to examine and direct its reasoning. In summary it permits a user to formulate and investigate models where the alternatives have been worked out, and the rationales of each decision explicitly laid out.

5.1. Status of Project

PLANET is written in Franz Lisp, HOUSE (Quayle, 1983), and RUP (McAllester, 1982) and runs on a VAX11/780. For the types of examples described in this paper, it requires approximately 2 megabytes of main memory. PLANET currently incorporates a limited amount of knowledge about the CMC manufacturing environment. A full-blown implementation is likely to require one or two man-years of effort, mostly by CMC personnel familiar with the various aspects of the planning problem. However, since most of the reasoning machinery is in place, much of this should be a largely a mechanical exercise.

The meta-model described in this paper has been entirely "hand crafted", that is, assembled by us on the basis of domain specific knowledge extracted by us from CMC documents and personnel. Space limitations do not permit discussion of that part of the meta-model which is capable of defining new alternatives, categorizing them according to the relevant bases, and synthesizing new heuristic rules. However, this part of the program has not been extensively tested -- a realistic assessment of the program's extensibility will be meaningful only after a sizable corpus of domain-specific knowledge has been encoded. Nevertheless, with its potential extensibility the PLANET framework offers an inherently rich environment within which to build a performance system that can incorporate, maintain, and reason about large amounts of knowledge across various areas and levels of an organization.

REFERENCES

Davis, Randall., Interactive Transfer of Expertise -- Acquisition of new inference rules, *Artificial Intelligence,* No.4, 1979.

Davis, Randall., TEIRESIAS: Applications of Meta Level Knowledge, in *Building Knowledge Based Systems in Artificial Intelligence* (Davis and Lenat, eds), McGraw-Hill, 1982.

Dearborn, D.C. and Simon, H.A., Selective Perception: A Note on the Departmental Identifications of Executives, *Sociometry*, volume 21, 1958.

de Kleer, J., Doyle, J., Steele, G. and Sussman, G., AMORD : Explicit Control of Reasoning, Proceedings of the Symposium on Artificial Intelligence and Programming Languages, 1977.

Doyle, Jon., A Truth Maintenance System, AI Laboratory Memo 521, MIT, 1978.

Dhar, Vasant., PLANET: An Intelligent Decision Support System for the Formulation and Investigation of Formal Planning Models, Ph.D Dissertation, University of Pittsburgh, 1984.

Goldberg, A. and Robson, D., *SMALLTALK-80: The Language and its Implementation,* Addison-Wesley: Reading, Mass 1983.

Hewitt, Carl., Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence*, 8, 1977, pp. 323-364.

McAllester, D., Reasoning Utility Package, AI Laboratory Memo 667, April 1982.

Moon, David. and Weinreb, Daniel., Lisp Machine Manual, MIT, 1981.

Nilsson, Nils., *Principles of Artificial Intelligence,* Tioga, 1981.

Pople, Harry, E., Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics, *Artificial Intelligence in Medicine*, Peter Szolovits (ed), Westview Press, Boulder, Colorado, 1982.

Quayle, Casey., Object Oriented Programming in Franz Lisp, Working Document, Decision Systems Laboratory, University of Pittsburgh.

Reitman, Walter., Applying Artificial Intelligence to Decision Support: Where do Good Alternatives Come From? in *Decision Support Systems*, Ginzberg, Reitman and Stohr (eds), North-Holland, 1982.

Simon, H.A., *The New Science of Management Decision*, Prentice-Hall, 1977.

Stallman, Richard. and Sussman, Gerald., Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, Artificial Intelligence, volume 9, No.2, October 1977, pp 135-196.