RULE-BASED VERSUS STRUCTURE-BASED MODELS FOR EXPLAINING AND GENERATING EXPERT BEHAVIOR

Vasant Dhar
Department of Information Systems
Graduate School of Business Administration
New York University

and

Harry E. Pople
Decision Systems Laboratory
University of Pittsburgh

March 1987

Center for Research on Information Systems
Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

IS-87-27

This paper appears in the *Communications of the ACM*, June 1987

RULE-BASED VERSUS STRUCTURE-BASED MODELS FOR EXPLAINING AND GENERATING EXPERT BEHAVIOR

Vasant Dhar
Department of Information Systems, New York University

Harry E. Pople
Decision Systems Laboratory, University of Pittsburgh

This paper appears in the *Communications of the ACM, September 1987.*

We would like to acknowledge Marilyn Stelzner and anonymous referees for their comments and suggestions which have contributed significantly in sharpening the presentation of this paper.

Abstract

Flexible representations are required in order to understand and generate expert behavior. While production rules with quantifiers can encode experiential knowledge, they often have assumptions implicit in them, making them brittle in problem scenarios where these assumptions do not hold. Qualitative models achieve flexibility by representing the domain entities and their interrelationships explicitly. However, in problem domains where assumptions underlying such models change periodically, it is necessary to be able to synthesize and maintain qualitative models in response to the changing assumptions. In this paper, we argue for a representation that contains partial model components that are synthesized into qualitative models containing entities and relationships relevant to the domain. The model components can be replaced and rearranged in response to changes in the task environment. We have found this "model constructor" to be useful in synthesizing models that explain and generate expert behavior, and have explored its ability to support decision-making in the problem domain of business resource planning, where reasoning is based on models that evolve in response to changing external conditions or internal policies.

1. Introduction

A major decision for an expert system builder is one of how expert knowledge is to be represented. The representational framework adopted can play an important role in how subsequent data are obtained, interpreted, and assimilated into the framework. Some researchers argue that expert knowledge is best encoded as rules. Others argue that qualitative models are a more accurate and/or robust representation of such knowledge.

In this paper, we argue for a more fundamental unifying representation, of which rules and qualitative models are particular expressions. The primitive elements of the underlying representation are model components which can be synthesized to compose interpretations that explain observed behavior, and exhibit expert behavior in problem solving situations. Rules can then be viewed as summaries of model behavior, summaries which may derive from assumptions about unknown conditions in the problem domain.

The representation we describe is based on the results of a collaborative research project with a computer manufacturing company aimed at understanding the problem of business resource planning, so that a computer-based model might be designed to support the planning problem. In attempting to understand the reasoning processes of experienced planners, we constructed several computer models to simulate expert behavior. The earlier models, which were "run" on real cases against experts, were rule-based, developed in OPS5 (Forgy, 1981) and then AMORD (de Kleer et. al, 1977). The brittleness of such models led us to investigate alternative, qualitative model representations -- which served as the point of departure for the existing scheme, implemented as a computer model called PLANET (Dhar, 1984).

In a manufacturing environment, the object of interest is a manufacturing system which is a structural arrangement of activities with flows of materials, designed to produce certain outputs while taking cognizance of resource constraints such as capital, space and labor. In the *planning* stages, the structure is *hypothetical*, based on assumptions about the future. Given the uncertainties involved, a important part of a planner's job centers around the dynamics of the structure. The dynamics are of two types. The first involves reasoning about parametric changes (i.e. what happens if the output needs to be doubled) with a given structure. The second involves dealing with assumption changes in the task environment, which can require *structural* modifications to the model in order to achieve the desired goals, given certain resource constraints.

2. Representations for understanding and generating expert behavior

During the earlier phases of this investigation, we attempted to model expertise involved in planning as rules. This turned out to be inadequate for a variety of reasons, which we discuss below. This discussion also motivates the need for an alternative representation scheme which we describe in the remainder of the paper.

2.1. Production Rules With Quantifiers and Certainty Factors

It has been argued that the basic advantage of the rule based representation scheme is the modularity rules provide (Forgy and McDermott, 1977; Davis and King, 1977) and the simple uniform interpretative procedure that is often sufficient in rule based systems (Duda and Shortliffe, 1985).

Within the rule-based paradigm, the probabilistic approach has been commonly used for modeling uncertain knowledge (Shortliffe, 1976; Duda et.al, 1979). As an example of a propositional rule incorporating probabilistic information, consider the following example from a manufacturing context:

```
IF:   1.  There is increased throughput of material
THEN: 2.  It is likely (0.8) that additional labor is required.
```

Here, the numeric weight (0.8) is intended to express the rule-author's degree of belief in the consequent, given that the antecedent is shown to be true. Such numeric weights are often referred to as "certainty factors", after Shortliffe (1976). More recently, Heckerman (1986) has argued that these factors can be interpreted as a special case of probabilities.

2.2. Hidden assumptions in production rule models

While this type of knowledge structure provides a good *general* idea about the relationships among certain variables of a problem, there is a danger associated with asking an expert to articulate knowledge in this form. If forced, the expert might specify such a rule and certainty numbers. However this is likely to be accompanied by appropriate qualifications and/or remarks to the effect that such a rule is difficult to express in the abstract, that is, it is *conditional* on various circumstances. For example, if pressed, the manufacturing plant expert would say that extra labor might not be required if the work rules could be changed, or if the plant were laid out more efficiently, or if part of the production could be offloaded to another plant, etc. Massaging all such qualifications into a numeric estimate may be possible, but of questionable value in that it is likely to force out important contextual knowledge surrounding the rule. Yet, this contextual knowledge may play an important role in the actual *reasoning* process of the expert in a problem solving situation.

2.3. Making Exceptions Explicit

An alternative approach toward modeling uncertainty, which has its origins in Stallman and Sussman's dependency directed reasoning paradigm, is the "reasoned assumptions" approach of Doyle (1983) which handles the treatment of uncertainty in non-statistical terms while leaving the propositional information unchanged. A discussion of the details is beyond the scope of this paper and can be found in Doyle (1983). The basic difference is that conditionals of a problem situation (defaults or exceptions to general propositions) are recognized explicitly instead of being "homogenized" into certainty scores as in the probabilistic approach. For example, a rule that takes explicit cognizance of exceptions and/or defaults might be

"IF there is an increase in throughput, THEN increase direct labor UNLESS you offload part of the manufacturing process to another facility, or UNLESS ...."

As Doyle points out, eliciting knowledge in this manner requires carrying the knowledge acquisition a step further:

> "One cannot simply reformulate probabilistic rules as reasons according to their certainty factors. To re-express a database of expertise, we require the knowledge acquisition process carried a bit further than usual. The approach of reasoned assumptions supposes that numerical judgements of certainty often hide more specific information not yet made explicit by the expert informant. When the expert says "IF A, THEN it is likely (0.3) that C," this really means that many exceptional cases are familiar to the expert. One might ask the informant to list these exceptions as a set B, in order to qualify the rule by writing it as A || B ||- C (which means *A without B gives C* or *conclude every sentence in C if every sentence in A has been concluded and no sentence in B has been concluded*), but it is often as difficult to think of exceptions offhand as it is to think of ordinary heuristic rules. Instead, we apply the same technique to articulating expertise as that already practiced, namely the informant expresses what is clear, and then formulates and reformulates the missing cases, exceptions, and generalizations by repeatedly examining the system's performance on test problems. At bottom, we always have rules of the form "Usually, IF A, THEN C" or "Usually, IF A, THEN $\sim$C," which we express as defeasible or default reasons, and we express the intermediate degrees of uncertainty by case analysis and reasons stating exceptions to generalities." (Doyle, 1983. Parenthesized explanation added.)

To summarize, the non probabilistic approach encourages eliciting information about conditionalities explicitly. Doyle's argument is that via repetitive case analyses, it should be possible to get the expert to articulate the exceptions and defaults instead of "losing information" via an implicit translation of these into numerical scores. Implicit to the argument is the position that defaults are knowable in principle as opposed to unknowable because of randomness in the phenomenon being investigated.

Doyle's analysis assumes that the "knowledge engineer" will ultimately converge on the right set of rules along with their associated exceptions and defaults. However, this approach sacrifices parsimony in representation by opening up the the possibility of admitting large numbers of defaults and exceptions,

leading to large numbers of rules which can become particularly difficult to keep up-to-date in problem domains characterized by a changing reality. Many exceptions can arise in situations with a significant number of degrees of freedom. For example, if the throughput rule mentioned in the last section has as its underlying context a complex model of the manufacturing process with a large number of interrelated entities, the effects of increased throughput could be assessed in many different ways, each depending on the conditions or constraints attached to the various parts of the manufacturing process. For such problems, rather than try to explicate all the defaults and exceptions, it is more fruitful to think in terms of a more parsimonious model that represents explicitly, the primitive entities and relationships of interest. Under changing conditions, the model must be maintained to reflect the changing reality.

3. Generating detailed behavior via qualitative models of manufacturing

Several detailed accounts of qualitative representations, all in domains involving simple physical systems, appear in a special issue of the AI Journal (December 1984). Because of the recency of such work and the diversity of the ontological primitives used in characterizing the domains, there is no standard set of conventions or notation for describing qualitative models. The basic approach, however, can be characterized as follows:

> "...in order to understand a physical system, the description of a system's behavior must be derivable from the *structure* of the system. The term structure refers to the components of the analysis, component behaviors, and the connections between components." (Bobrow, p.1, 1984, italics added).

Understanding the behavior of such a system in response to a change requires starting at the point where the change is induced, and propagating its effects through the connections. Depending on the task that the system is addressing, the behavior that results can be used for purposes such as prediction/simulation (Forbus, 1984), envisioning (de Kleer and Brown, 1984), and diagnosis (Davis 1984; Genesereth, 1984).

Our concerns center around two kinds of changes, described at the outset of the paper: the first involve a simulation within an existing model structure whereas the second type involve restructuring the model itself. The latter functionality calls for synthesizing qualitative models from model parts represented at various levels of abstraction. Before considering how these changes can be accomodated, let us consider how a qualitative model in the manufacturing context is synthesized in the first place.

3.1. Model synthesis from partial model components

Assume that some computer manufacturing process is 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. Each of these areas involves performing certain broad *functions*

which in turn involve several lower level activities. There is a certain logic for organizing activities in various configurations. For example, if a module has been assembled, it makes sense to pass it through an activity that tests it for faults (typically open and short circuits) that might have been induced as a consequence of the assembly process. In effect, knowing that assembled modules have to be processed establishes the need for a test activity, and at the same time establishes constraints on where assembled modules can be directed in the overall manufacturing process. Let us denote the activity that tests for faults as *Manufacturing Induced Faults Test* (MIF-Test). Following such a test, it is necessary to ensure that each component of the module is *functional*, that is, performs within tolerance limits. We call this *Functional test* or F-test for short. Finally, since functional modules must perform satisfactorily as a whole under conditions they will have to endure in the finished product, another testing stage can be expected. This testing, sometimes referred to as "testing modules at speed," we refer to simply as *Speed test* (S-Test). Thus, the overall module-check function could involve the *configuration* of activities shown in Figure 1. Such an arrangement can be considered a "typical" arrangement of activities in the same way a molecule is a typical structural arrangement of atoms. There can be several typical arrangements of activities corresponding to a function.

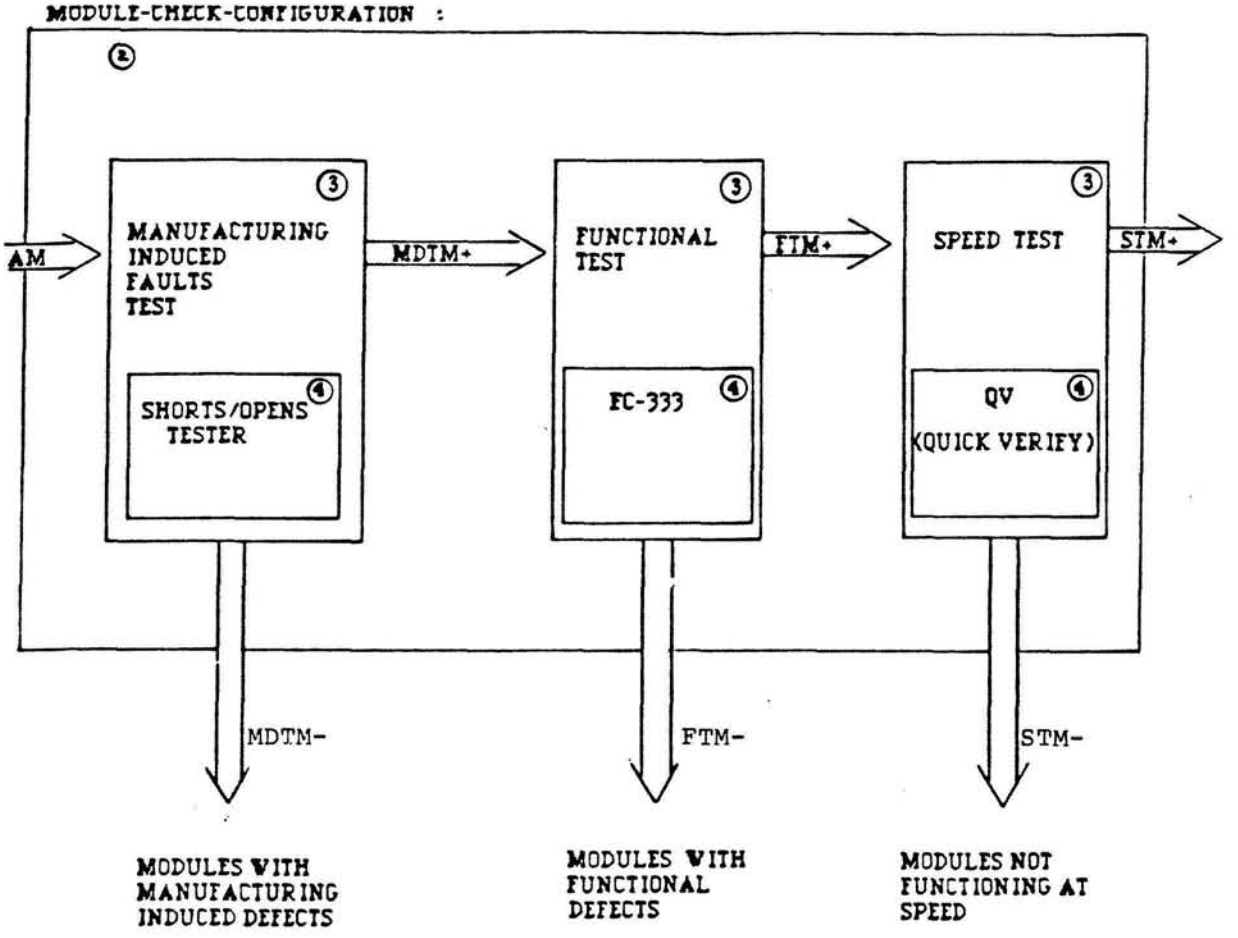
In laying out the rationale behind the configuration outlined above, we have in effect performed a synthesis of components (in this case, activities) in arriving at a coherent, more comprehensive conceptualization. This conceptualization denotes part of the overall qualitative model (which we also term the *synthetic* model), which can be composed by integrating other model components into increasingly elaborated "task complexes" (Pople, 1982). A visual representation of the qualitative model then begins to resemble that shown in figure 2. The connections among the activities are material flows, which connect various parts of the model. An activity can have multiple inputs and outputs. Also, certain materials can be processed by several activities, as indicated in table 1. In the planning context, the qualitative model such as that in figure 2 is one among many that might have been synthesized. This is because the synthesis task is inherently underconstrained, requiring consideration of sets of alternatives in the various parts of the task environment and making choices from among them. This flexibility in what selections will be made in various parts of the task environment makes it possible to have many structural arrangements, one of which in schematized in figure 2. The nested boxes in figure 1 and 2 show the various model components. Each of the components of the model can be conceptualized as an instance of an object which is defined in terms of a set of properties. For example, the method object (labeled 4 in the figures) can be defined as the tuple

<id, i/o, direct-labor, indirect-labor, capital, space>

where the first property designates a method's identity, i/o is a set of algebraic input/output relationships among the volumes of materials flowing through the activity, and the next four are numerical amounts of

FIGURE 1

A schematic of a configuration of activities that comprise the module-check function.

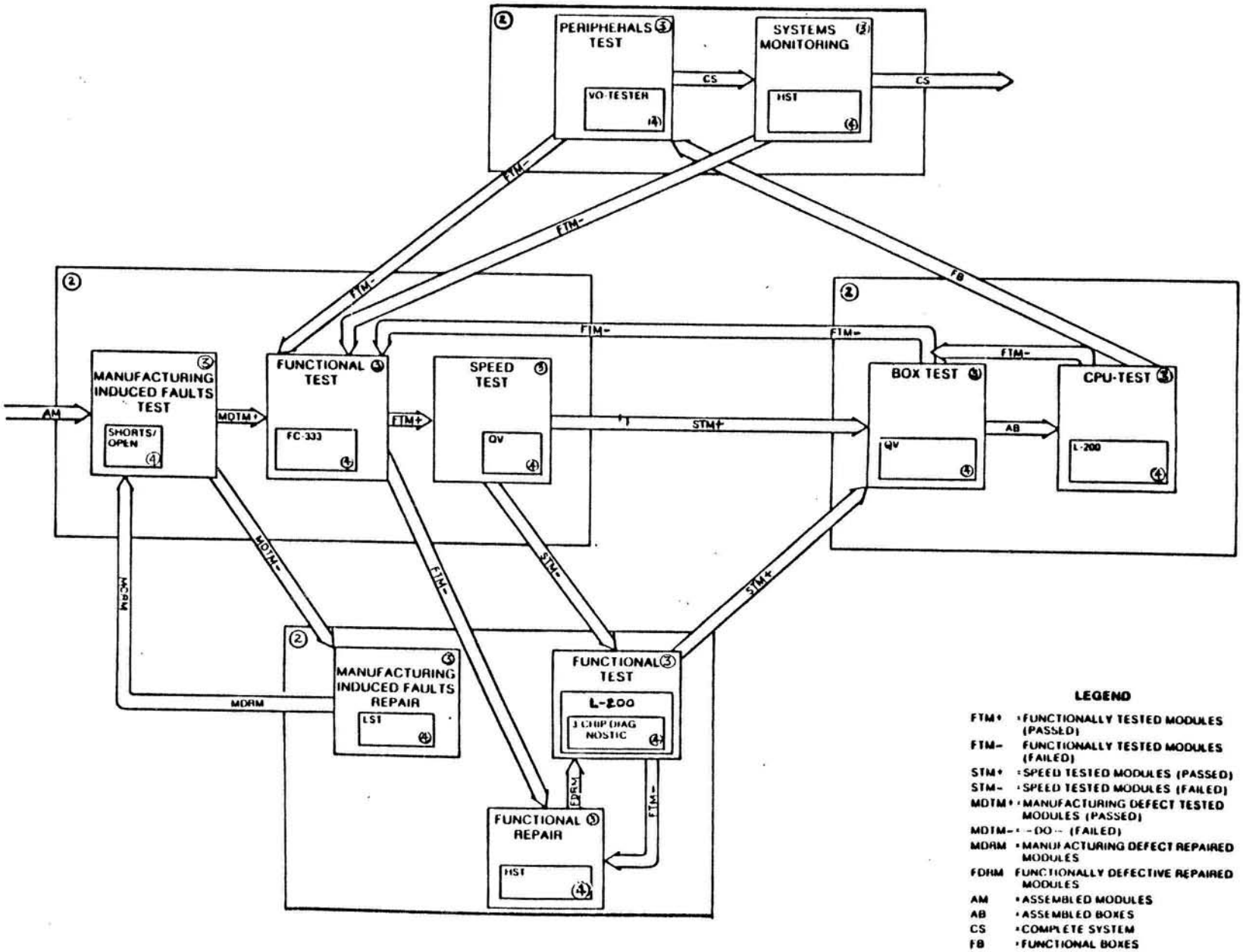


LEGEND

AM	ASSEMBLED MODULES
MDTM+	MODULES PASSING MIF TEST
MDTM-	MODULES FAILING MIF TEST
FTM+	MODULES PASSING FUNCTIONAL TEST
FTM-	MODULES FAILING FUNCTIONAL TEST
STM+	MODULES PASSING SPEED TEST
STM-	MODULES FAILING SPEED TEST

FIGURE 2

A segment showing various functions (and lower level model entities) and the connections among them.



resources (in appropriate units) or algebraic relationships for calculating resource requirements, expressed in terms of the material flows through the activity. An activity (labeled 3 in the figures) can be described by the tuple

<id, choice-set, method, inputs, outputs>

which consists of properties indicating the identity of an activity, the set of potential methods that can be used, the method chosen, the inputs, and the outputs to the activity. The configuration and function objects are described in terms of similar properties which we illustrate shortly. In summary, instances of functions, configurations, activities, and methods constitute sets of *primitive model components*, subsets of which can be selected in synthesizing the complete model.

Functions –> Materials	module-check	module-repair	kernel-check
FTM-	(F-test S-test)	(F-repair)	x
STM-	x	(F-test F-repair)	x
MDTM+	(F-test S-test)	x	x
STM+	x	x	(Box-test)
AB	x	x	(Box-test CPU-test)

Table 1

This table indicates activities (lists inside the cells) in the different functions (on the horizontal axis) that can process the various materials (on the vertical axis). The materials and activities are from figure 2. Each of the activities in the cells has an associated "transformation function" (defined in the text) that indicates the relationships among their inputs and outputs.

3.2. Reasoning using the synthesized model

Let us assume that an interpretative model has been assembled by means of a model building program (described in the next section), and that the current state of the computer model includes components indicated in figure 2.

Reasoning about the "what if volume is doubled" situation using such a model becomes somewhat different than in the rule-based model described previously. To see how, consider an activity such as F-

Test in module-check (which we refer to as module-check:F-Test) as being represented in terms of a structured object data structure that incorporates knowledge about the method employed and the types and proportions of material flows through it, as follows:

```
{activity
 id: module-check:F-Test
 method: FC-333
 inputs:  ((MDTM+ 110 module-check:MIF-Test))
 outputs: ((FTM+ 100 module-check:S-Test)(FTM- 10 module-repair:F-Repair))
}
```

where the inputs and outputs are lists of triples indicating the type of material, the amount, and source (of input) or destination (of output). The numbers are the results of solving certain algebraic input/output relationships that are specifiable in the data structure corresponding to its method object. Resource requirements for the activity also depend on the method used, in this example, the requirements of the FC-333 method (illustrated shortly). A different choice of method for this activity might entail a different set of resource requirements.

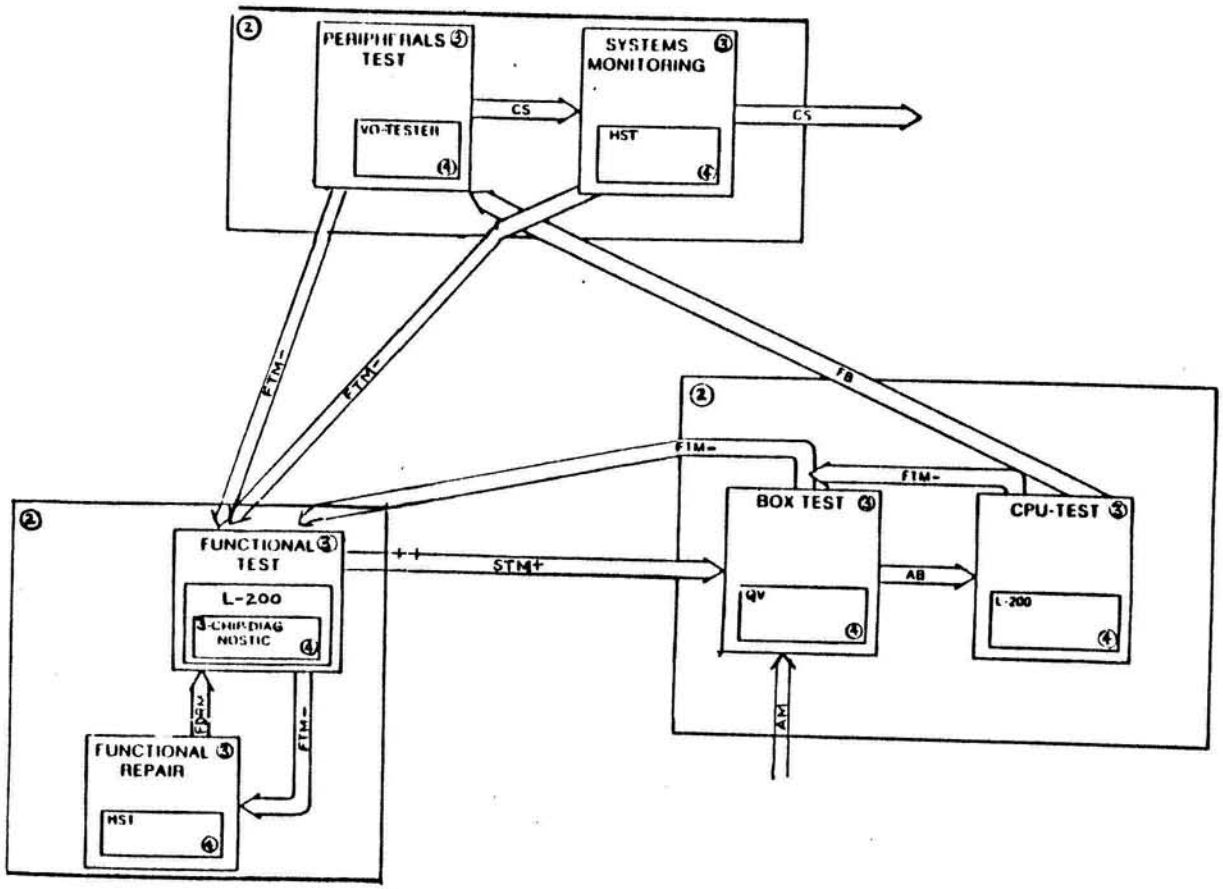
Assessing the impacts of doubling the volume requires running a simulation initiated at the activity where an input or output is changed, with the resulting changes (assuming for the moment that proportions of materials through the activity are maintained) being propagated to connected activities. For example, in a model corresponding to figure 2, if the input material to the activity module-check:MIF-test is doubled, its outputs to the two activities it is connected to, namely module-check:F-Test and module-repair:MIF-Repair increases in the proportion determined by certain input/output relationships. These increased flows carry forward to all related activities. In the small model fragment of figure 2, this would involve recomputing the increased flows for each of the ten activities. The corresponding resource requirements can then be computed for the revised flows. The results of this process may or may not differ from those of the rule-based model depending on the assumptions about the manufacturing process embodied in that model. Specifically, the accuracy of the rule-based model would depend on the correctness of its hidden assumptions for this particular problem scenario.

The effect of changing certain types of assumptions underlying the qualitative model can be more dramatic. For example, if there were an organizational policy change to offload part of the module production to another facility, there might be no need for the module-check function at all. This requires a structural modification -- removing certain components of the model and their dependencies and establishing new connections among a new set[1] of model components. For such a change, part of the relevant model might appear as in figure 3, which has no module-check function. In our system, this type of restructuring (i.e. the decision not to have module-check) is initiated by the user, with the system assisting with the maintenance or establishment of appropriate dependency relationships among the

[1] In this example, the new set is a subset of the old one. The new set can also be larger.

FIGURE 3

A schematic showing some structural modifications to the model segment of figure 2.



LEGEND

FTM+	• FUNCTIONALLY TESTED MODULES (PASSED)
FTM–	FUNCTIONALLY TESTED MODULES (FAILED)
STM+	• SPEED TESTED MODULES (PASSED)
STM–	• SPEED TESTED MODULES (FAILED)
MDTM+	• MANUFACTURING DEFECT TESTED MODULES (PASSED)
MDTM–	• -DO- (FAILED)
MDRM	• MANUFACTURING DEFECT REPAIRED MODULES
FDRM	FUNCTIONALLY DEFECTIVE REPAIRED MODULES
AM	• ASSEMBLED MODULES
AB	• ASSEMBLED BOXES
CS	• COMPLETE SYSTEM
FB	• FUNCTIONAL BOXES

various components of the model.

The results of running the simulation on this model are likely to be different from those of the model corresponding to figure 2; obviously, no additional labor or other resources would be required for assembling and checking modules as in the previous situation because the module-check activities no longer exist. Resource requirements for the other activities can be computed in the same way as described above. If the expert were to try to articulate this situation in terms of rules, he might be forced to modify the previous ones, or worse still, provide new rules to handle the qualitatively different model corresponding to the changed problem scenario. The important point to note is that if we consider an evolving structure of the type we have described as an underlying reality and rules being summary descriptions of the behavior of such a system, it is not surprising to find experts articulating different rules for seemingly identical problem situations. For a knowledge engineer attempting to model the expert knowledge as rules, converging on the "right" set of rules can become virtually impossible.

In concluding this section, we should point out that we do not regard rules as being inherently limited in their representational ability compared to other types of representations. Rather, rule-based models frequently compile domain knowledge into heuristics. Encouraging the expert to articulate the "rules of the domain" can have the effect of forcing out important contextual knowledge surrounding the rule, leading to differences between the behavior of the system and the expert on real cases. In the case of the planning problem, we found the rules articulated by the expert to be summary descriptions of reasoning based on a more detailed model of the task environment that involved explicit knowledge about primitive entities and relationships in the task environment. What appeared as exceptions to rules (leading to more rules) were easily understood in terms of a structural representation of the task environment, which we describe next.

4. Synthesizing and managing the qualitative model

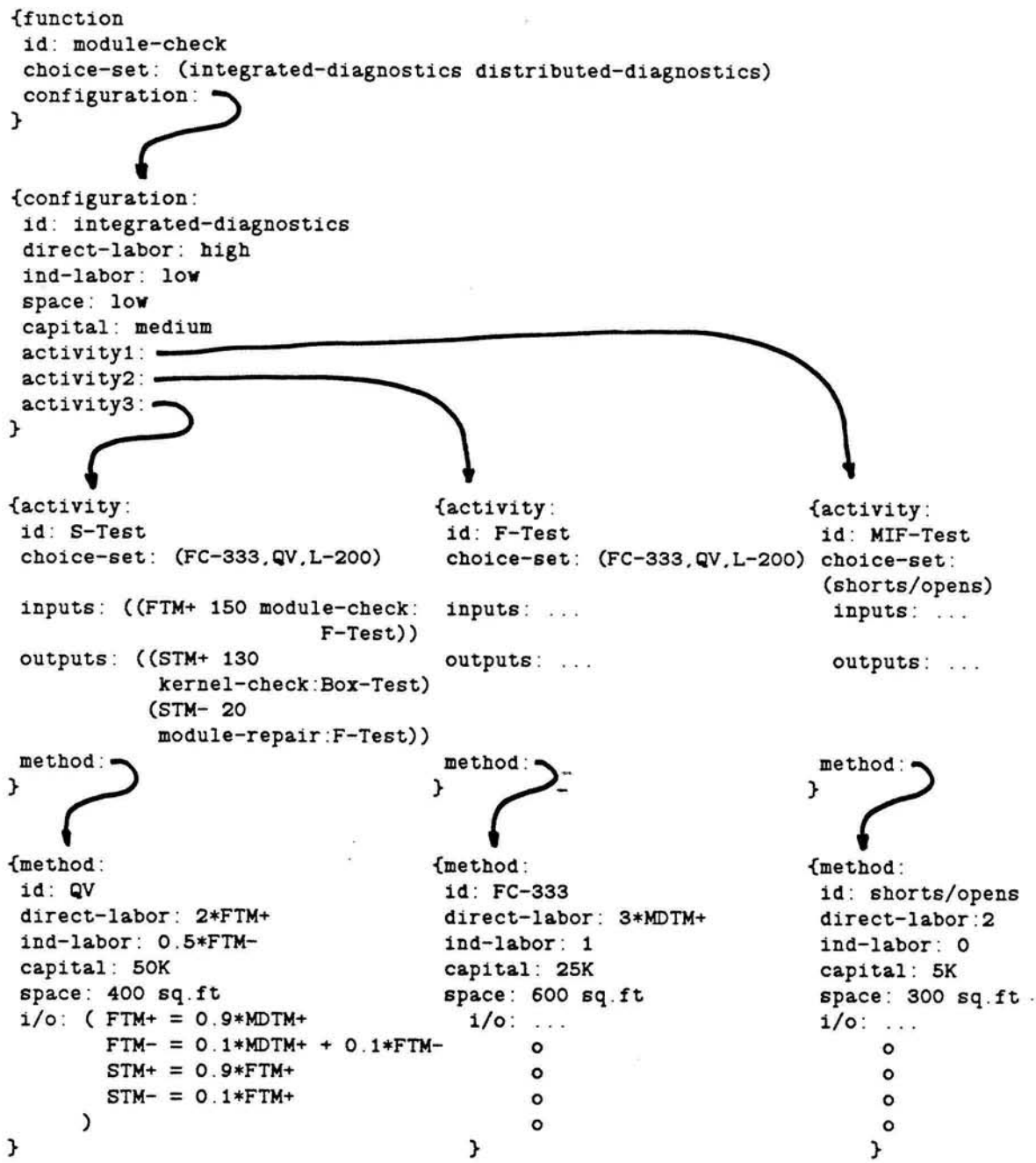
The qualitative model we have described encodes several types of knowledge. Its fundamental building blocks are the model components and the relationships among them, and a procedural component that threads the components into complete models (or modifies existing models in response to changes).

4.1. Model components as a hierarchy of objects

In the model synthesis program implemented by Dhar (1984, 1986), synthesis is a design task that involves making choices from among competing alternatives (called choice sets) in different parts of the task environment. Each alternative is represented as an instance of a structured object data type. The instance variables of the object type correspond to the attributes used to characterize the object.

Each object corresponds to a certain level of abstraction. As indicated in figures 1 and 2, there are four levels at which choices are made. These levels are shown in figure 4 where each entity at the bottom end of a line corresponds to an alternative with respect to the entity at the top end of the line. In effect, each line represents a *part-of* relationship. When a model is synthesized, an object can be chosen as part of a higher level object connected to it. Arrows in figure 4 indicate material flows (corresponding to the double arrows of figures 1,2 and 3).

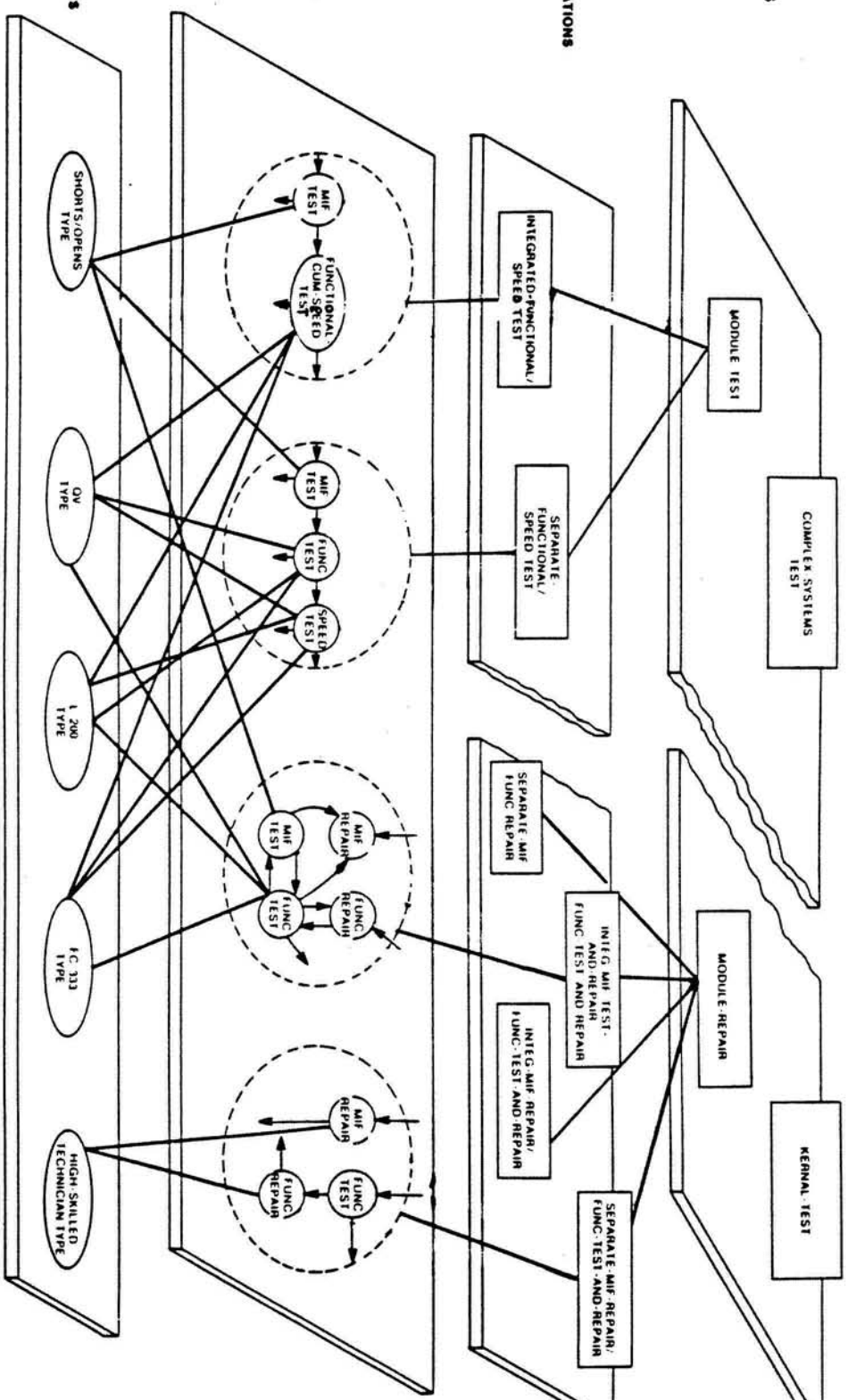
The structure and role of the objects can be described by considering the model segment of figure 2, which is an instantiation of an object hierarchy, such as the one in figure 4. The model segment is a top-down "projection"of this hierarchy. For example, the module-check function within this model segment is represented as follows:



The structured object descriptions are enclosed within braces. The top level object, in this case the module-check function, contains the choice set of alternative configurations. When a configuration is selected, it becomes part of the function object (indicated by the pointer). The configuration contains qualitative descriptions of resource requirements (the role of which will be described shortly), and activity objects that are part of it. When a method is selected for an activity, an object corresponding to this method becomes part of the activity. The method objects contain specific resource requirements, inputs that they can process, and the relationships between their inputs and outputs.

FIGURE 4

A hierarchy of model fragments (objects) from which the "model constructor" synthesizes alternative models.



A complete qualitative model is composed of several such top-down instantiations corresponding to different parts of the task environment. Each choice set involves one or more *part-of* relationship between an object and lower level objects, one of which can become included in it; in effect, in a completely formulated model, one such relationship is instantiated for each choice set relevant to the task environment. In the remainder of this section, we describe how these choices become part of the synthetic model.

4.2. Constraints

There are two types of constraints that must be considered in synthesizing the model. The first type are qualitative constraints, which express dependency relationships between alternatives in the different choice sets. Structurally, such constraints are similar to production rules. In our system, a qualitative constraint is represented as a two part list structure. The left hand side consists of a set (list) of identifiers of choice set alternatives. Each identifier is of the form *obj:alt-id* where *obj* is an object (which may contain lower level objects as choices) containing the alternative identified by *alt-id*. The right hand side consists of a single negated or non-negated identifier. For example, the constraint "((module-check:F-Test:FC-333) –> module-check:S-Test:~FC-333)" states that the method FC-333 (the *alt-id* of the FC-333 object) cannot be employed simultaneously by the F-Test and S-Test activities in the module-check function.

A choice set alternative identifier *obj:alt-id* in a qualitative constraint is satisfied if the alternative identified by *alt-id* has been selected in object *obj*. If the identifier is negated, then it is satisfied if some other alternative has been selected in the object *obj*. We say that a qualitative constraint is satisfied as long as it is not violated. A violation occurs when each item in the left hand side of the constraint is satisfied and the right hand side is not satisfied.

The second type of constraint is the quantitative constraint which affects choice somewhat more indirectly. Simple quantitative constraints are expressed in the form "<*alg-exp rel-op alg-exp*>" where *alg-exp* is an algebraic expression defined over problem variables (these variables designate resources such as labor, capital, or space or can be exogenous variables defined by a user) and *rel-op* is a relational operator. Simple quantitative constraints can be combined into more complex ones using the boolean operators AND, OR, and NOT. Each quantitative constraint is a boolean function that returns the value *false* when the constraint is violated and *true* otherwise.

In our system, each quantitative constraint is translated into a form consisting of a predicate followed by forms corresponding to the algebraic expressions. For example, the following expression in Lisp-like prefix notation states that the capital requirements of the F-Test activity in module-check is less than $100K: "(less (capital module-check:F-Test) $100K)" where *capital* designates a function that returns the value of the capital required by its argument, in this case the F-Test activity. *less* is a predicate that

returns false if the value of its first form is greater than that of its second form, and true otherwise.

4.3. Inter-object communication in model synthesis

Objects have associated "message types" defined over them.[2] By representing design components as objects capable of sending and receiving messages, the responsibility for assimilating a change becomes distributed. For example, if as a consequence of a qualitative constraint it is determined that FC-333 method should be established as a choice in some activity as a consequence of a qualitative constraint, a message is dispatched to that activity object which is responsible for establishing it as the choice and performing any associated book-keeping. Similarly, the responsibility for effecting a high level change such as that involving offloading module-checking to another facility is accomplished by dispatching a message to the module-check object to eliminate itself, which also removes the lower level objects within it.

The qualitative model can be synthesized from scratch or modified. If it is designed from scratch, the choice process proceeds in a top-down fashion. If a modification is required, parts of the model are first removed along with their dependencies (explained shortly), and revised choices made in these parts of the model. In making these choices, the system alternates between two modes of operation which, following Stefik (1980), we term *constrained* and *heuristic* modes. In the constrained mode, qualitative constraints determine (or preclude) choices from specified choice sets. The heuristic mode becomes operative when propagation of choices comes to a halt. In this situation, a choice must be made from some choice set in order for problem solving to continue. The choice can be specified directly by the user, or made automatically by the system based on an evaluation function defined over the choice set attributes, that is, the resource categories. The order in which choice sets are examined is based on a prioritization of the choice sets which reflects the user's view of the importance associated with the decisions corresponding to them.

If the evaluation of alternatives within a choice set is made across low level alternatives where detailed resource requirements are available, a numeric evaluation function is used. If the comparison involves the more abstract pieces of the model where only qualitative values are available (such as high, medium or low), the evaluation is heuristic, involving combination of these values into overall ratings.

It can be the case that a violation of either a qualitative or quantitative constraint occurs before the model is fully formulated. If a constraint violation occurs while the system is in the constrained mode, it becomes necessary to revise a previously made choice in some choice set, and then to continue forward

---

[2] In object oriented programming, these are often referred to as "methods". Since we use the term method to denote a model object as opposed to a procedure, we refer to the procedural knowledge component as a message type.

from that point. Specifically, a heuristic mode choice and its dependents are discarded. A revised choice is then made from the choice set containing that last heuristic mode choice and problem solving continues. The process of determining the most appropriate "culprit" choice set, which is not of central interest here, is described in Dhar (1986) and Dhar and Quayle (1985).

If a constraint violation occurs while in the heuristic mode as a consequence of a choice made from a choice set, a new choice is made from that choice set if possible. However, if all choices from a choice set have been tried unsuccessfully, then, as in the case above, a previous heuristic mode choice is discarded, and a revised choice made from the choice set containing that heuristic mode choice.

4.4. Maintaining dependencies among choices

Whenever a choice is made, a two part structure called a node is created. The first part of the structure consists of the identifier of the choice, and the second part consists of the reasons for making the choice (similar to Doyle's "set of support" (Doyle, 1978)). If the choice is made as the consequence of a constraint relationship becoming satisfied, the node corresponding to the choice has a "deduction justification", which consists of identifiers of other choices (the left hand side of a constraint relationship). When made heuristically, the justification for the choice is the preference function that was used in making the choice; these heuristic choices become candidates for retraction in backtracking situations. Deductively established choices can be retracted only when all their supporting choices are retracted.

The primary use of the dependency network is to enable incremental modifications to the qualitative model. If a change requires some existing choice to be retracted (i.e. "no FC-333"), the choice is removed from whatever support sets it appears in. Choices whose supports have become empty are then retracted, and the deductive-heuristic choice cycle described above comes into play, resulting a modified qualitative model. In this way, the qualitative model always reflects the current set of choices made from different parts of the task environment, and the dependency network reflects the relationships among the choices.

4.5. Roles of the synthetic model

In summarizing this section, it is worth restating the motivations that have shaped the model synthesis system described above, and the methodological considerations that a knowledge engineer must be cognizant of in trying to explicate a model of expertise.

In attempting to understand the decision-making processes of the expert analyzed in this research, we began in a classical knowledge engineering fashion where the expert was encouraged to articulate the "rules" about the domain. Several prototype systems based of these rules were run on real cases against the expert. These systems were found to be inadequate in matching the behavior of experts who often

resorted to reasoning based on deeper domain knowledge in explaining or rationalizing a case. This led us to postulate that the deeper context-specific models were being synthesized on the fly from finer grained knowledge components, and that this synthetic model was then used to rationalize the data, that is, the data were interpreted to "fit" the model.

A second observation during the initial knowledge acquisition exercise was that different experts, or the same expert at different times in the knowledge acquisition process, articulated different rules about identical problem scenarios. Such an observation might cause a knowledge engineer to speculate whether the expert is being inconsistent. While this is possible, we found that the different rules often derived from differences in the assumptions, that is, the different contexts underlying these rules. In the planning context, where decisions change often, a decision maker may have considerable latitude in the assumptions (choices) introduced into the synthetic model, leading to different rules being articulated in similar cases. In such cases, rules may be adduced as being summary descriptions of behavior, where the structure of the model underlying the behavior may not be apparent.

In summary, based on the initial knowledge acquisition exercise and subsequent case analyses, we were drawn toward exploring the reasons for the divergence between the rule models articulated by the expert, and the deeper reasoning process that seemed to emerge with real case data. What gradually became apparent was that the hidden assumptions that typically went unarticulated during knowledge acquisition, often played a central role in case analyses. Further, because of these hidden assumptions, it often appeared that different rules were being articulated on different occasions for identical problem scenarios. Trying to identify the complete set of exceptions and defaults for every rule was cumbersome and yielded too many rules that were difficult to maintain. This experience made it clear that reasoning about planning situations was really based on *synthetic* evolutionary models that are highly sensitive to problem context. This led us to believe that the challenging part of the decision maker's job is formulation, that is, maintaining the integrity of the model under changing conditions so that it can be used for reasoning. The model synthesis system we have designed has been shaped by these concerns.

5. Practical uses of the model synthesis system

Since a pragmatic goal of this research is to provide planning managers with a knowledge based support tool, it is worth summarizing some of the uses of our modeling system for decision support, as well as its potential for supporting other aspects of group modeling situations in large organizations.

The process of model building in large organizations involves a necessary diffusion of responsibility among several individuals or groups within the organization. When a large problem is decomposed in this way, it is the responsibility of the groups to generate feasible alternatives with respect to the part of the

task environment for which they are responsible. It is then the job of a coordinator to synthesize some of these alternatives into a coherent whole, keeping in mind the relationships among the different parts of the task environment and other constraints such as those on resources. In synthesizing the whole, only a small subset of the alternatives actually considered make their way into the whole. In effect, the modeling exercise involves a successive elimination of the degrees of freedom that characterize the initially open-ended problem situation. Simon (1973) provides a general description of this type of modeling in the context of architectural design.

The system we have designed corresponds to parts of the above characterization of the modeling process. Specifically, once the alternatives with respect to various decisions have been generated, the system can synthesize alternative models in collaboration with a user, while taking cognizance of the problem constraints. By preserving the sets of alternatives generated, the system can also serve as a repository of the *process knowledge* that is associated with the modeling exercise. Although the system can not derive new alternatives with respect to the choice points, it does allow for modification of choice sets and constraint relationships, and can change the models in light of this evolving knowledge. In summary, because of its ability to bring into the fore previously foreclosed options or newly specified ones, the system provides a window into the inherently underconstrained problem situation out of which the specific models (instantiations) are synthesized.

The system can also be useful for situations in which groups of people are involved in modeling because of its potential to enhance communication and coordination across the multiple parties involved in the modeling exercise. Specifically, it can make explicit the impacts of decision changes in one part of the task environment on other parts. For example, it can highlight to a high level manager or coordinator, the repercussions of local changes in choices on the overall model -- an activity which is currently time consuming and prone to error. By using the system as an intelligent assistant that is familiar with all parts of the model, the user has the ability to explore variations of the model that might not be practical to investigate otherwise.

In conclusion, our view of the role of models for explaining expert behavior and for management decision support is that it is unwise to expend great time and effort attempting to engineer the "correct model" when in fact such a model might be tentative. We regard the model as a tentative structure assembled in a specific context from the underlying primitives of the task domain. Although further evidence is needed to generalize our framework to other types of planning problems, our observations in this research have led us to believe that a central concern of decision makers in problems such as planning is one of preserving the faithfulness of models under changing conditions, particularly when the projections generated from them involve large amounts of resources. If we recognize that much of the use of models in business organizations occurs in changing scenarios, it is necessary to have systems that

synthesize models that reflect the changing reality. This problem of "getting the model right" and maintaining it in light of changing conditions is an important responsibility of a manager or his support staff that can benefit greatly from knowledge based support. The model formulation and synthesis system described here can undertake some of this responsibility, effecting a balanced division of responsibility between the decision-maker and the system.

REFERENCES

Bobrow, D., Qualitative Reasoning about Physical Systems: An Introduction, *Artificial Intelligence*, vol. 24, nos 1-3, 1984.

Davis, R., Diagnostic Reasoning Based on Structure and Behavior, *Artificial Intelligence*, vol. 24, nos 1-3, 1984.

Davis, R., and King, J., An Overview of Production Systems, in *Machine Intelligence, 8*, Elcock and Michie (eds), John Wiley and Sons, New York, 1977.

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.

de Kleer, J., and Brown, J.S., A Qualitative Physics Based on Confluences, *Artificial Intelligence*, vol. 24, nos 1-3, 1984.

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

Dhar, V., and Quayle, C., An Approach to Dependency Directed Backtracking Using Domain Specific Knowledge, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August 1985.

Dhar, V., Using Knowledge Generated in Heuristic Search for Non-Chronological Backtracking, *Journal of Computational Intelligence*, volume 2, number 3, 1986.

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

Duda, R., and Shortliffe, E., Expert Systems Research, *Science*, January 1985.

Duda, R.O., Gashnig, J., & Hart, P., A Computer-Based System for Mineral Exploration in *Experts Systems in the Microelectronic Age* by Michie (ed), Edinburgh Press, 1979.

Forbus, K., Qualitative Process Theory, *Artificial Intelligence*, vol. 24, nos 1-3, 1984.

Forgy, L., OPS5 Users Manual, Technical Report, Department of Computer Science, Carnegie-Mellon University, 1981.

Forgy, L., and McDermott, J., OPS: A Domain-Independent Production System, Proceedings of the Fifth International Conference on Artificial Intelligence, 1977.

Genesereth, M.R., The Use of Design Descriptions in Automated Diagnosis, *Artificial Intelligence*, vol. 24, nos 1-3, 1984.

Heckerman, D., Probabilistic Interpretation for MYCIN's Certainty Factors, in *Uncertainty in Artificial Intelligence*, Lemmer, J.F. and Kanal, L (eds), North Holland, 1986.

McDermott, J., R1: A Rule-Based Configurer of Computer Systems, *Artificial Intelligence*, vol 19, no. 1, 1982.

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.

Shortliffe, E., *MYCIN: Computer Based Medical Consultation*, American Elsevier, 1976.

Simon, Herbert., The Structure of Ill-Structured Problems, *Artificial Intelligence*, volume 4, number 3, 1973.

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.

Stefik, Mark., Planning With Constraints, Ph.D Thesis, Department of Computer Science, Stanford University, 1980.