Overview of ROME: A Reason-Oriented Modeling Environment

Donald W. Kosy and Ben P. Wise

CMU-RI-TR-85-21

Intelligent Systems Laboratory
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

December 1985

To appear in *Artificial Intelligence in Economics and Management*, L. F. Pau (ed.), North-Holland Publishing Co. (in press). This work was sponsored by The Robotics Institute and the Eastern Electronics Systems Company.

Table of Contents

List of Figures

Abstract

The ROME system is an experimental decision support system generator that incorporates facilities originally developed for "expert" artificial intelligence systems. Like other DSS generators, it provides a set of tools for building, running, and documenting mathematical models of business activities. Beyond that, however, the system includes explicit differentiation between hard facts and softer information such as assumptions and expectations, a set of procedures for evaluating and explaining results, and a natural language user interface. In this paper, we discuss the representation of knowledge in ROME and show how it is used to answer questions, explain trends, detect anomalous values, and assess the credibility of model assumptions.

1. INTRODUCTION

In recent years, it has become very much easier to build computerized planning models for use in business decision making. Systems embodying such models have come to be called "decision support systems" (DSSs) and by using a decision support system generator [10], one can construct models in a few hours or days that would have taken weeks or months otherwise. At the same time, artificial intelligence (AI) techniques have emerged from the laboratory which have proven to be quite effective in solving certain classes of real-world problems. It has therefore been suggested [5,9,11] that there may be considerable value in combining these technologies to yield more powerful DSSs than are currently available.

The ROME system is an experimental DSS generator which has been built to investigate the application of AI techniques to quantitative business analysis. Its name is an acronym which is meant to characterize the approach we have taken. As a "modeling environment," ROME is like other DSS generators in that it provides a set of tools for building, running, and documenting mathematical models of business activities. A hierarchically structured database is part of that environment and an English-like command interface is used to access both the tools and the data. More importantly, however, ROME is "reason-oriented" in that it includes explicit differentiation between hard facts and softer information, such as assumptions and expectations, and procedures for evaluating and explaining the results it produces. A major goal of our research to date has been to realize a system that can explain why a result should be believed or give reasons why it should be challenged.

2. BACKGROUND: THE PLAN REVIEW PROBLEM

Our interest in evaluation and explanation was originally motivated by a study [7] of long range resource planning at a large U.S. computer firm, which we will call EESCO. A *resource plan* is essentially a quantitative projection of the amounts of resources that will be needed to produce the volume of goods a firm expects to sell in future years. The major resources in EESCO's plan, for example, were labor, factory floor space, raw material, facilities, capital equipment, and expense funds. Projections of resource needs are used by a firm to allocate resources currently held and to make decisions on acquiring new ones.

Creating resource projections involves a combination of calculation and judgment that can be represented in a quantitative planning model with judgmental parameter values. Although it may be large, this model is simply a set quantitative relationships among variables, expressed by formulas and conditional statements, together with a set of input data. The formulas encode such things as accounting relationships between price and cost, aggregation of resource subtotals into totals, and the relative amounts of each type of resource required to produce a unit of product. The data include product parameters, manufacturing process parameters, and forecasts of resource price levels. Judgment is involved in the forecasts and in characterizing the resource demands of new products and processes.

Since there are many ways to design a manufacturing process for a given product, planners will normally choose a design that is responsive to various *goals* the firm has set for itself. EESCO's goals, for example, included increases in productivity, improved customer service, and maintenance of a stable workforce in spite of substantial changes in product types and volumes. These goals affect the specification of planned processes and these specifications, in turn, are reflected in the planning model parameters.

As in many large firms, the overall planning task at EESCO was decomposed into a hierarchy of subtasks corresponding to the different levels of the organizational hierarchy. As projections at lower levels were generated, they were reviewed and then consolidated into projections for the parent level. If approved at the parent level, these projections were frozen into final plans to be submitted to the next level up. For example, if several products were manufactured in the same plant, a plan for each product would be developed, reviewed at the product level, consolidated into a plant plan, reviewed at that level, and then, if approved, submitted for consolidation at division level.

At all levels, the resource plans themselves were displayed as arrays of numbers where the columns specified planning periods (e.g., years) and the rows showed the projections for each type of resource for each period. The labor section of a hypothetical plan for one of EESCO's plants is shown below.

=======Marysville==========FY82	====FY83	====FY84	====FY85	====FY86	====FY87
d1	95	127	85	38	46	54
i1	68	87	69	49	51	51
year end people	163	214	154	87	97	106
ave people		189	184	121	92	102

Figure 1: Part of a Resource Plan for EESCO's Marysville Plant

The review steps in the planning process turned out to be very important to its success. Reviews were performed by planning managers who were ultimately held accountable for the plans they submitted to the next level up. They recognized that a plan is only as good as the assumptions it is based on and so one purpose of their review was to identify and assess those assumptions. In particular, reviewers wanted to make sure that assumptions used at lower levels matched their own assumptions about how the firm operated so that there would be no inconsistencies in combining lower level results into an aggregate. If an inconsistency was detected, an explanation would be sought from the lower level planner.

They also recognized that for a plan to be acceptable at the next higher level, it must be responsive to organizational goals. At EESCO, goals for one level did not necessarily have to be met by all lower levels. However, it was important that they be met by the combination of lower level values in the aggregate. Hence, a second purpose of the review was to evaluate aggregate results against goals and, if there were a mismatch, to negotiate revisions in lower level plans that would eliminate it.

While there are a number of DSS generators that could have been used to build and run the planning models at EESCO, and one, in fact, was being used, none of them provides much help for the review process. In particular, none distinguishes assumed relationships from necessary ones, none can really explain the results produced, and none has any facilities for evaluating results against established goals or norms. The ROME project was initiated to investigate what might be done to fill this gap.

3. THE ROME SYSTEM

The approach we have taken in ROME is to provide a modeling system in which results can be evaluated and explained by comparative analysis. More specifically, results can be evaluated by comparing them to expectations, and results can be explained by comparing derivations of

differences. These operations are facilitated by a database structure that allows inheritance of symbolic information and a user interface that allows natural expression of expectations and questions.

As shown in the figure below, the system itself consists of four major components: a database, an interface module called ROMULUS, an explanation component called ERGO, and an evaluation component called REMUS.

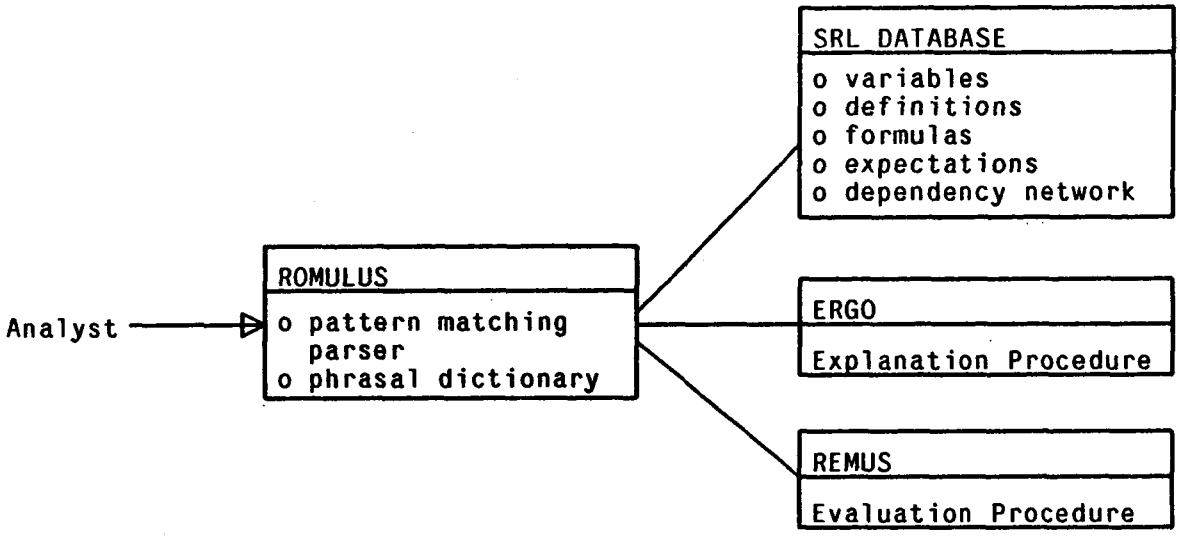

Figure 2: ROME Components

In the following sections, we describe each of these components in turn.

4. THE DATABASE

The database contains all the non-procedural knowledge ROME uses and is organized around the set of variables comprising a planning model. The variables are arranged in a hierarchical network and are represented in the database by records called *schemata* using the frame-style language SRL [14]. This hierarchy represents a categorization of variables into classes, subclasses and instances according to their level of specificity. The figure below shows part of the tree of variables for the resource plan shown in Figure 1.

At the top are the different types of variables. At the next level, a variable is differentiated into subclasses according to the real-world entity it relates to, such as a particular product, plant, company, or whatever. We call such entities *anchors*. At the bottom level, each instance of a variable is associated with a *column* which further specializes its context of evaluation by some other entity, such as a time period. The bottom level is thus equivalent to a set of two dimensional arrays where each element is indexed by anchor and column.

Unlike an array, however, the *is-a* and *instance* links in the tree are used to propagate information from higher to lower levels by "inheritance." In SRL, information about a schema is represented by

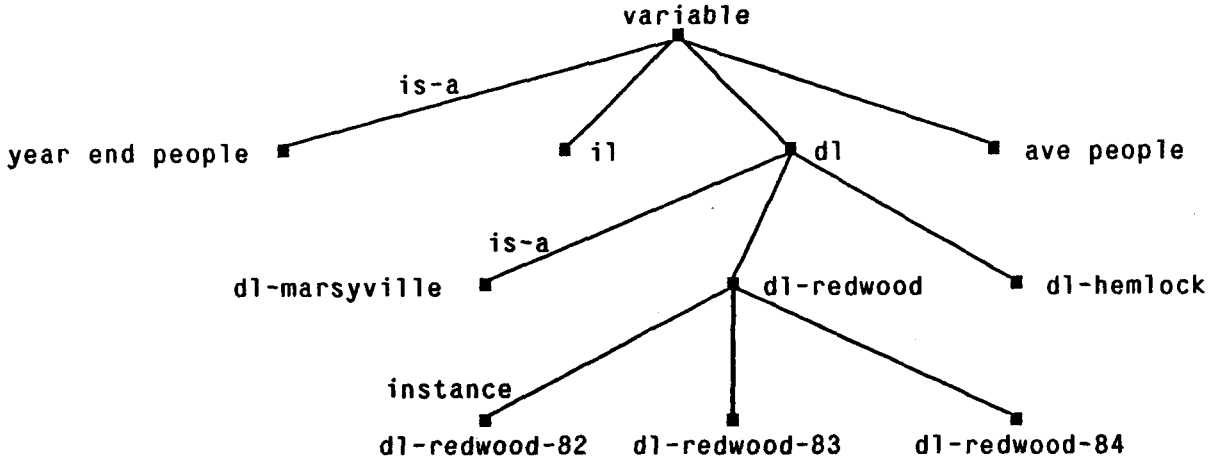

Figure 3: Part of a Tree of Variables

storing symbolic values in *slots* attached to the schema. Information is said to be inherited when a slot in one schema is automatically filled with a value taken from another schema where the schemata are related by a chain of *is-a* and *instance* links. Thus, in ROME, if information about a variable is stored at the class level, such as a formula for computing its value, all subclasses and instances of that variable will inherit this same formula. Furthermore, there is no restriction on making one class of variable a subclass of another class so that information about the latter may be inherited by the former. For instance, IL and DL might be made subclasses of the class "labor" and would then inherit information from this more abstract level.

Properties of variables may be linked into the tree at any level, and lower level properties normally supersede higher level ones. For instance, a DL formula for product 2 will be the same for every period unless it is overridden by a different formula at the instance level. However, the inheritance procedure may also be directed to augment lower level properties with properties from higher levels. This kind of inheritance is used for expectations and goals.

The content of the database represents the knowledge ROME has about variables, values, formulas, and the real-world activities they describe. Properties of variables include their definition (verbal or arithmetic), their numerical values in different contexts, the source of those values (input or formula), and whether the variable is taken to be endogenous or exogeneous to the system being modeled. Input variables may be further described as 'controllable' if the decision-maker using the model can change (or negotiate a change in) its value in the real world.

Values have a modality property which indicates whether they are empirical facts, estimates based on other variables, or simply assumptions. Likewise, formulas are categorized as being either definitions, estimation equations, or assumptions. Definitions are exact equalities, e.g., 'sales = volume • selling price'. Estimation equations, on the other hand, are not exact and are used to characterize a process without actually simulating or defining it. Estimates of endogenous variables can be viewed as approximations of their values while estimates of exongenous variables can be viewed as predictions. Examples of estimation formulas include regression equations, trend

extrapolation, use of growth-rate factors, etc. All other formulas that are not definitions or estimations are considered to be assumptions.

Knowledge about what a variable describes can be represented in the database in two ways. First, actual values the variable has had in the past can be stored as facts. Second, expected or desired values can be specified by expressions that denote norms or goals, respectively. A *norm* in ROME is a relationship among variables that "should" hold true under "normal" circumstances according to experts in the domain of the model. A *goal* is a statement of an organizational objective or policy that can be expressed in terms of model variables. Norms and goals are used by REMUS to evaluate model results and, as part of that process, may be used to infer formulas for variables that are needed for the evaluation but are not explicitly present in the model.

To keep track of derived information, a dependency network [2] is used to link properties to their sources. A dependency is a record that contains the justification for a slot's value in terms of the action that produced the value plus the inputs to that action. Examples of actions that set values are communication with the user, execution of a formula, and inheritance from a higher level. Inputs corresponding to these actions are the content of user input, values of variables contained in a formula, and the is-a/instance chain that allowed inheritance, respectively. Using these dependencies, the system can tell, for example, what variables affect what other variables and this, in turn, provides the information ERGO needs to analyze and explain model results.

5. ROMULUS

ROMULUS stands for Reason-Oriented Modeling Using a Language Understanding System and this component acts as the intermediary between the user and the other components. Its responsibilities include initial processing of user inputs, maintenance of the database, and control of the output display. As a "service module" for the rest of the system, our main goal for ROMULUS has been that it be unobtrusive: It should be fast, flexible, forgiving, informative, and easy to learn without training.

Like other natural language interfaces [6,12], ROMULUS is designed to simulate understanding in a limited domain by recognizing all likely forms of expression for a limited number of ideas. This recognition is performed at the sentence level by a pattern-matching parser [1] in conjunction with a large number of grammar rules. Every grammar rule is a triple consisting of a lefthand side, a righthand side, and an arrow between them. A pattern to be matched is expressed by a sequence of expressions comprising terminal symbols, non-terminal symbols and operator expressions. Some sample grammar rules (for a 'why' question) are shown below:

```
(why <aux> <focus-item> <change> (!setting := ?<adjunct>) ?%qmark)
=>
(why-change !tense !focus !change !setting)

        <aux> -> ((!tense := (&i present (do | does | is | are))) |
                  (!tense := (&i past (did | was | were))))
        <focus-item> -> (!focus := (&u <change>))
        <change> -> ((!change := change) | go (!change := (up | down)) |
                     (!change := (&i up (increase | rise)))|
                     (!change := (&i down (decrease | fall))))
        <adjunct> -> ((in | for | at) (+ ~%qmark))
```

Non-terminal symbols are marked by angle brackets, operator expressions by parentheses, and

terminals are left unadorned. Variables to be bound to constituent strings when a pattern is matched are indicated by an exclamation mark.

Rather than continuing with a formal description, it is easier to see how a rule works by tracing through its application. For example, if the first word in an input sentence is *why*, the parser will try to apply the first rule above (and all other rules whose first pattern element can match the terminal symbol *why*). If the next word matches the pattern specified by <aux> (for auxiliary verb), the variable **!tense** will be set. A word will match <aux> if it is one of the terminals *do*, *does*, *is*, or *are*, in which case **!tense** is bound to *present*, or if it is one of *did*, *was*, or *were*, in which case **!tense** is bound to *past*. Next, the non-terminal <focus-item> can be matched by any string of words that is followed by a string that matches <change>. If this process can be continued until all of the input is matched, the righthand side of the top level rule will be executed using the variable bindings made during the match. For instance, if the input is *Why did Marysville year end people go up in 83*, the routine 'why-change' will be called with bindings **!tense** = *(present)*, **!focus-item** = *(Marysville year end people)*, **!change** = *(up)*, and **!setting** = *(in 83)*.

The next task is to find referents for the phrases that have been extracted from the input. A referent is a schema in the database such as a variable, anchor, or column, and the association is made by a phrase interpreter using a phrasal dictionary. Unlike a typical natural language system, wherein the vocabulary is fixed, the ROMULUS interpreter must be able to deal with whatever terminology the user happens to employ. Hence, it does not attempt to determine what a phrase means, but only what it refers to. Interpretation proceeds by (1) segmenting a phrase into parts using syntactic cues, (2) determining the referent of each part, and (3) assembling a retrieval request for the whole phrase based on the semantic type of each referent in it. This procedure is flexible enough to recognize that

Marysville year end people in FY83
year end people at Marysville in 83
yep(mr) in 83
year end people(83,mr)

all refer to the same variable, if the dictionary contains appropriate entries.

The dictionary entries for words and phrases are simply pointers to whatever schemas they refer to. This makes it easy to represent synonyms, abbreviations, and acronyms as just other names for the same schema. Moreover, since dictionary entries can be phrases, the interpreter can almost always guess the appropriate correction for a misspelled word in a phrase by examining the other words around it and finding the phrase that matches most closely. The algorithm used for spelling correction is taken from [4].

In general, the interface to a modeling system must accept three sorts of inputs: questions, commands, and declarations. The current ROMULUS grammar includes interrogative forms for asking questions about variables, getting advice on functions the system can perform, and asking why results came out the way they did. It includes imperative forms for controlling the display and requesting evaluations. The remaining forms specify the syntax of declaratives which are used to define variables, express formulas, establish goals and norms, assign values to input variables, and declare all other properties of variables. Examples of declarations related to the variables in Figure 1 are shown in Figure 4, where hemlock and redwood are the names of two products manufactured at Marysville:

```
Define dl to be the number of in-house direct labor EESCO employees at year end
Define il to be the number of in-house indirect labor EESCO employees at year end
Define year end people to be dl + il
Define ave people to be the average number of plant employees during a year
Estimate ave people(y) to be (year end people(y) + year end people(y-1))/2
Estimate dl to be roundup(dl-hrs * 1000 / hrs/shift/yr)
Declare marysville, redwood, and hemlock to be anchors
Use mr to refer to marysville
Use rw to refer to redwood
Use hm to refer to hemlock
Define dl(mr) to be dl(rw) + dl(hm)
Define il to be plant mgmt + product il
Define dl-hrs to be std hrs + n-std hrs
Declare fy82 values to be factual
Let std hrs(rw) = 0 44.40 73.60 50.79 62.68 77.21
Let std hrs(hm) = 85.12 91.13 22.56 0 0 0
```

Figure 4: Definitions and Declarations

System output is displayed on a split screen. The top portion shows a spreadsheet display of model results very much like that shown in Figure 1. Each display column is headed by a column name (fiscal years in Figure 1) and the name of the anchor for the values being displayed is shown at the upper left. Such displays are produced by a *show* command and any calculations are performed when they are needed to show the results. The bottom portion is used to display user inputs, answers to questions, explanations, and the results of REMUS's evaluations. Thus, the top portion is a report of results and the bottom portion shows the dialogue that has occurred so far.

The overall effect is to give the user the impression that he is conversing with an analyst about the results shown in the report. After the declarations shown above have been processed, the conversation below exemplifies the kinds of questions ROMULUS can answer.

```
> What does "dl" mean?
It stands for direct labor.

> How is direct labor defined?
That represents the number of in-house direct labor EESCO employees at year end.

> How is it calculated?
In general, the formula is:
        dl = roundup(dl-hrs * 1000 / hrs/shift/yr)      [estimate]
```

Note that pronouns, such as the word *it*, are allowed to refer back to previously mentioned items. Determining the referent of a pronoun is easy in this environment because interrogative and imperative forms in English clearly establish a focus of interest. Output generation is also easy because everything the system can say can be cast in the form of output templates to be instantiated as needed. Although all the techniques ROMULUS uses are simple, combining them with a large number of patterns (grammatical, phrasal, etc.) allows the interface to be much more flexible and forgiving than in other modeling systems.

6. ERGO

ERGO stands for Explaining Results Generated by Others and its function is to answer 'why' questions about numbers. In asking a 'why' question, a person is expressing a desire to have something clarified that he doesn't understand. In the case of numbers, what needs to be clarified is how the numbers were derived and why the derivation produces the results observed.

We can divide results to be explained into two categories. The first category comprises *explicit results*, which are those that appear explicitly in the output of a model and are produced directly by formulas. ERGO takes a very simple and direct approach to explaining explicit results: it simply displays the formula and the values of subordinate variables. For example, ERGO's answer to the question *Why does dl(mr) equal 128 in 83?* in our example model would be that *dl(rw) = 56 and dl(hm) = 72 and dl(mr) = dl(rw) + dl(hm).* Thus, a 'why' question about a variable's value is treated as a 'how' question about its derivation, and the answer is simply the derivation itself. This approach is very similar to that taken by MYCIN [3], which explains its conclusions by exhibiting the rules used and the certainty factors of subordinate clauses. There are a number of similarities between the formulas in a model and the rules in an expert system, and so the explanations can be similar as well. Although simple, this technique is important because it saves the user from having to search the model himself to find out where a result came from.

The second, more interesting, category comprises *implicit results*, which are those that come from comparisons the user makes between values. Implicit results are referenced in questions like *why does dl(mr) go up in 83?* and *why is there a dip in year end people in 85?*. Answering such questions involves explaining differences. For example, the difference to be explained for a direction question such as *Why does dl(mr) go up in 83?* is the difference between the value of the variable in the given context, (mr,83), and its value in the previous context, (mr,82). We call the first value the *focus value* and the second the *referent value*.

The procedure ERGO uses to explain differences is based on comparing derivations. To be comparable, the derivations must involve the same formula, say g, so that the difference, Δy, comes from evaluating g in the focus context vs the referent context:

$$\Delta y = g(a_f, b_f, c_f, \ldots) - g(a_r, b_r, c_r \ldots)$$

The subscripts on the arguments denote the two different contexts. We will let S denote the set of variables referenced in g: $S = \{a, b, c, \ldots\}$. Since all ROME variables are two dimensional, one can easily specify comparisons across columns, across anchors, or across both, by asking questions that specify the appropriate indices. For instance, to ask why DL for the redwood product is greater than DL for hemlock, the question form would be: *Why is dl(rw) greater than dl(hm) in 83?*. The comparison in this case would be between variables subscripted (rw,83) and variables subscripted (hm,83) in the formula that computes the value of DL. The formulas for the different instances of DL would be the same because they both would be inherited from the general formula for DL.

The first step in the explanation procedure is to find a set of variables X, where $X \subseteq S$, which is sufficient to account for Δy. To do that, we define a measure of significance, $\varepsilon(X,y)$, which measures the *effect* of variables in X on y in the focus context relative to the referent context. The definition is:

$$\varepsilon(X,y) \triangleq y_f - g(Z)$$

where the vector Z contains values of variables in X evaluated in the referent context and values for all other variables in S evaluated in the focus context. By successively choosing values of X to be first individual variables in S, then pairs, then triples, and so on, the derivation of the value of $\varepsilon(X,y)$ is made closer and closer to the derivation of Δy. When the derivations are close enough that the value of $\varepsilon(X,y)$ is a substantial fraction (80%) of the total difference Δy, we conclude that the effect of the variables in X is large enough to explain the difference. Variables in S but not in X are taken to be insignificant with respect to this difference.

Given the set of explanatory variables X, the second step is to express the explanation in words. In general, the answer will include (1) the differences that account for Δy, (2) the formula g, (3) identification of the primary explanatory variable(s) and (4) a qualification, which may express counteracting, reinforcing, or insignificant effects. Like the responses ROMULUS produces, ERGO's answers are generated from templates that are just sufficient to express in English the information that must be conveyed. To illustrate, if the input question is *Why did dl(mr) go up in 83?*, the answer would be:

> *Dl(mr) goes up in 83 because dl(rw) goes up and dl(mr) = dl(rw) + dl(hm). Although dl(hm) also changes, its effect was insignificant. Would you like me to continue?*

The explanation can be continued all the way down to the lowest level of the model, i.e., its exogenous variables. A more complete description and justification of both steps of this procedure is given in [8].

7. REMUS

The purpose of REMUS is to Review and Evaluate a Model's Underlying Structure. Of all ROME components, the design of this one has been most strongly influenced by the resource plan review problem at EESCO. First, it uses an encoding of evaluation criteria that is independent of the model itself. The reason for this is simply that such criteria depend much more on the aims of the reviewer than on the model he is reviewing. In ROME, evaluation criteria are expressed by statements of goals and norms. Second, it was the case at EESCO that some important criteria involved variables that were not explicitly shown on the resource plan. However, their values could be inferred from variables that were shown plus the reviewer's knowledge of the domain of the model. For this reason, ROME allows criteria to be phrased in terms of 'generic variables', if desired, which a reviewer may link to variables in a specific model. Finally, it was also the case for some important variables that reviewers had no criteria. When this situation arose, reviewers would first search for related variables they could evaluate, perform the evaluation, and then assess the impact of their evaluation on the original variable. To capture this aspect of plan review, the REMUS review procedure capitalizes on the hierarchical structure of resource plans to search for evaluable variables at lower levels in the derivation tree if needed.

The main steps in this procedure are shown below. To evaluate a variable *v*:

1. If there are criteria for the values of *v*, apply them and state conclusions. Check equalities before inequalities.
2. Find variables that explain the trend using ERGO and the formula for *v*. If there is no formula but there is an equality norm, use that. Call this set of variables S.
3. If there are variables in the formula which have criteria but are not in S, add them to S.
4. Recursively evaluate the variables in S.

The main conclusions that may be drawn are as follows:

1. If criteria are met, REMUS says so.
2. If a goal is not met, REMUS calls it 'problematic'.
3. If there is a moderate difference (< 30%) between a value and a norm, it is 'OK'.
4. If there is a large (> 30%) difference between a value and a goal, it is 'extraordinary'. If there is a large deviation from a norm, it is 'odd'.
5. If the effect on *v* of a large deviation in a lower level variable can be determined, REMUS suggests that *v* may be 'too high' or 'too low'.

This procedure can be continued until it reaches exogenous variables. If the initial variable *v* is at the subclass level, then each instance in the column sequence is evaluated in turn. A fuller discussion may be found in [13].

To use this procedure, one must declare the evaluation criteria in the form of goals and norms. For example, one goal at EESCO was that new products should be manufactured more efficiently than old products, or at least no less efficiently. This goal may be expressed to ROME as:

```
Declare comparable product to be an anchor variable
Declare current year to be a column variable
We want Resource Used/Unit to be no more than Resource Used/Unit(comparable
        product,current year)
```

One such resource, for example, might be the amount of labor used per unit of output. In accounting, labor usage is measured in hours required, given some manufacturing process and a "standard" volume of output:

```
Define Standard Hours/Unit to be Total Standard Hours / Q-produced
Declare Standard Hours/Unit to be a measure of Resource Used/Unit
Expect Total Standard Hours to equal Standard Hours/Unit * Q-produced
Declare Standard Hours/Unit to be exogenous
```

Of course there may be many such criteria, but one will suffice for illustration.

Notice that the variables just mentioned are not specific to the model defined in Figure 4. Rather, they represent quantities that are more or less universal in manufacturing operations and the relationships between them are essentially generic. As a textual convention, we have distinguished these generic variables by capitalizing their names. Relationships among them are expressed by declaring expected equalities. If variables in a specific model are made instances of generic ones, the corresponding relationships will be inherited. This may be done as follows:

```
Declare std hrs to be a measure of Total Standard Hours
Declare build schedule to be a measure of Q-produced
Let comparable product(rw) = hemlock
Define current year to be 82
```

If a generic variable is not linked to any specific one it will be instantiated directly for any relevant anchors and columns. This will be the case here for Standard Hours/Unit.

To show how this all works, we close with a sample evaluation:

```
> Evaluate dl(mr)
There are no evaluation criteria for dl(mr).  However, dl(mr) goes up in 83 because
dl(rw) goes up and dl(mr) = dl(rw) + dl(hm).  Although dl(hm) also changed, its effect
was insignificant.  Would you like me to continue? Y

Dl(rw) goes up in 83 solely because dl-hrs(rw) goes up and dl = roundup(dl-hrs * 1000 /
hrs/shift/yr).  Would you like me to continue? Y

Dl-hrs(rw) goes up in 83 because std hrs(rw) and n-std hrs(rw) both go up and dl-hrs =
std hrs + n-std hrs.  The increase in dl-hrs was due primarily to the increase in std
hrs. Would you like me to continue? Y

By norm 1, we expect std hrs to equal Standard Hours/Unit * build schedule.
This appears to be so for rw in 83.  Would you like me to continue? Y

By goal 1, we want Standard Hours/Unit to be no more than Standard Hours/Unit(hm,82).
This appears to be so for rw in 83.  But we also find Standard Hours/Unit(rw) to be much
less than Standard Hours/Unit(hm,82) in 83.  This seems extraordinary.  So dl(mr) may
be too low.  Would you like me to continue?
```

If the user answers *yes* at this point, evaluation will continue with the next column. It might have continued with an explanation of the increase in n-std hrs(rw), but this is an input variable with no known equality relationships and so there was no way to continue from there.

8. CONCLUSIONS

It can be argued that current DSSs support decisions in three ways. First, they automate some of the clerical component of an analyst's activities thereby enabling him to explore a broader range of options than would otherwise be possible. Second, they provide structural frameworks for some types of decisions in the form of computational models. Third, within the context of a model, they provide a free-flowing interaction with the system thereby introducing the possibility of helping the analyst arrive at a better understanding of the problem.

The support they provide is limited, however. For one thing, their clerical function does not extend much beyond calculating and formatting. Not much attention is paid to input validation, assumption checking, or keeping track of intermediate results. Nor is there much in the way of mechanisms by which results could be explained or justified to users who were not familiar with what a particular model does. If a user is skeptical of the results he gets, it is up to him to search for reasons to believe or challenge them.

In ROME, we have attempted to go beyond these limits by adding several AI-inspired capabilities to a traditional DSS base. Using ROMULUS, for example, an analyst can ask questions about properties of variables other than their values, which would not be possible with a strictly numerical system. Using ERGO, he can get explanations of results. Using REMUS, he can review the results in light of assumptions and check them against evaluation criteria.

These capabilities have been implemented using two basic AI techniques: search and hierarchical knowledge representation. The comparative analysis that ERGO performs, for example, requires searching through combinations of variables to find a set that accounts for a difference. The evaluations made by REMUS require a search for variables that are related to the variable being evaluated. Hierarchical representation has been important in linking individual variables to more abstract or generic classes whose properties should be inherited by the individuals.

Given the framework we have set up in ROME, there seem to be many opportunities for further research. For one, both explanations and evaluations are currently limited to one context at a time, e.g., one column. But, given a set of explanations or evaluations, it might be possible for the system to produce summary statements from them, or generalizations, that would identify what was significant about a whole series of values. Second, the technique of comparative analysis might be extended to explaining the results of what-if scenarios, or to finding the causes of differences between the actual performance of a firm and what was planned or budgeted. Thus, it may be possible to provide much better support for budget-variance analysis, for instance, than is currently available. Finally, we are very interested in whether the techniques implemented in ROME are robust enough for practical use. While we have tested the system on a few small models, it remains to be seen whether the concepts and procedures will scale up to larger, more complex models where explanation and evaluation should be especially valuable. These are among the issues we are currently exploring.

REFERENCES

[1] Boggs, W.M, Carbonell, J.G., and Monarch, I., "The Dypar-I Tutorial and Reference Manual," Computer Science Department, Carnegie-Mellon University, 1984.

[2] Charniak, E., Riesbeck, C.K., and McDermott, D.V., "Data Dependencies," in *Artificial Intelligence Programming*, Hillsdale, N.J.: Lawrence Erlbaum Associates, 1980.

[3] Davis, R., *Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases*, PhD Thesis, Computer Science Department, Stanford University, July 1976.

[4] Durham, I., Lamb, D.A., Saxe, J.B., "Spelling Correction in User Interfaces," *CACM*, V. 26, No. 10, pp. 764-773, October 1982.

[5] Gorry, G. and Krumland, R., "Artificial Intelligence Research and Decision Support Systems," in *Building Decisions Support Systems*, J. Bennet (ed.), Addison-Wesley, 1982.

[6] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., and Slocum, J., "Developing A Natural Language Interface to Complex Data," *ACM Transactions on Database Systems*, V. 3, No. 2, pp. 105-147, June 1978.

[7] Kosy, D.W., and Dahr, V., "Knowledge-Based Support Systems for Long Range Planning," The Robotics Institute, Carnegie-Mellon University, 1983.

[8] Kosy, D.W., and Wise, B.P., "Self-Explanatory Financial Planning Models," *Proceedings of AAAI-84*, pp. 176-181, August 1984.

[9] Scott Morton, M.S., "Expert Support Systems -- The Next Generation of Decisions Support," *National Conference on Decision Support and Expert Systems*, George Washington University School of Government and Business Administration, Washington, D.C., April 1985.

[10] Sprague, R.H., Jr., "A Framework for the Development of Decision Support Systems," *MIS Quarterly* 4, December 1980, pp 1-26.

[11] Sprague, R.H., Jr., "Knowledge Based DSS: A Research Progress Report," *Transactions Second International Conference on Decision Support Systems*, G. Dickinson (ed.), June 1982.

[12] Waltz, D.L., "An English Language Question Answering System for a Large Relational Database," *CACM*, V. 21, No. 7, pp. 526-539, July 1978.

[13] Wise, B.P., and Kosy, D.W., "Model-Based Evaluation of Long-Range Resource Allocation Plans," The Robotics Institute, Carnegie-Mellon University, 1985.

[14] Wright, M., and Fox, M.S., "SRL/1.5 User Manual," The Robotics Institute, Carnegie-Mellon University, 1982.