MODEL DESIGN IN THE PROSPECTOR CONSULTANT SYSTEM FOR MINERAL EXPLORATION*

Richard Duda, John Gaschnig and Peter Hart

ABSTRACT
Prospector is a computer consultant system intended to aid geologists in evaluating the favorability of an exploration site or region for occurrences of ore deposits of particular types. Knowledge about a particular type of ore deposit is encoded in a computational model representing observable geological features and the relative significance thereof. We describe the form of models in Prospector, focussing on inference networks of geological assertions and the Bayesian propagation formalism used to represent the judgmental reasoning process of the economic geologist who serves as model designer. Following the initial design of a model, simple performance evaluation techniques are used to assess the extent to which the performance of the model reflects faithfully the intent of the model designer. These results identify specific portions of the model that might benefit from "fine tuning", and establish priorities for such revisions. This description of the Prospector system and the model design process serves to illustrate the process of transferring human expertise about a subjective domain into a mechanical realization.

I. INTRODUCTION

In an increasingly complex and specialized world, human expertise about diverse subjects spanning scientific, economic, social, and political issues plays an increasingly important role in the functioning of all kinds of organizations. Although computers have become indispensable tools in many endeavors, we continue to rely heavily on the human expert's ability to identify and synthesize diverse factors, to form judgments, evaluate alternatives, and make decisions – in sum, to apply his or her years of experience to the problem at hand. This is especially valid with regard to domains that are not easily amenable to precise scientific formulations, i.e., to domains in which experience and subjective judgment plays a major role.

The precious resource of human expertise is also a fragile and transient one: the departure of a crucial expert from an organization may cause serious dislocations; senior people impart their knowledge to younger colleagues, but the demand for their talents may not leave sufficient time for such educational efforts.

* This work was supported in part by the Office of Resource Analysis of the U.S. Geological Survey under Contract No. 14-08-0001-15985, and in part by the Nonrenewable Resources Section of the National Science Foundation under Grant AER77-04499. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of either the U.S. Geological Survey or the National Science Foundation.

During recent years research in the field of artificial intelligence has produced effective new techniques for representing empirical judgmental knowledge and using this knowledge in performing plausible reasoning. The best known application of these techniques has been in the area of medical diagnosis, where computer programs have achieved high levels of performance (Pople et al., 1975; Shortliffe, 1976; Weiss et al., 1977; Yu, 1978; Szolovits, 1976). Other applications include planning experiments in molecular genetics (Martin 1977) and monitoring instruments in intensive care units (Fagan 1978). This paper concerns a similar computer program, called Prospector, that is being developed to help geologists in exploring for hard-rock mineral deposits. The characteristic of plausible reasoning shared by the domains of medical diagnosis and mineral exploration is common, to some degree, to many other diverse evaluation tasks as well. Hence the purpose of this paper is to illustrate, by a case study for the domain of mineral exploration, the general process of capturing and encoding human expertise into a mechanical realization.

II. OVERVIEW OF THE PROSPECTOR SYSTEM

The Prospector system is intended to emulate the reasoning process of an experienced exploration geologist in assessing a given prospect site or region for its likelihood of containing an ore deposit of the type represented by the model he or she designed. Here we use the term "model" to refer to a body of knowledge about a particular domain of expertise that is encoded into the system and on which the system can act. The empirical knowledge contained in Prospector consists of a number of such specially encoded models of certain classes of ore deposits. These models are intended to represent the most authoritative and up-to-date information available about each deposit class.

In Prospector's normal interactive consultation mode, the user is assumed to have obtained some promising field data and is assumed to desire assistance in evaluating the prospect. Thus, the user begins by providing the program with a list of rocks and minerals observed, and by inputting other observations expressed in simple English sentences. The program matches these data against its models, requests additional information of potential value for arriving at more definite conclusions, and provides a summary of the findings. The user can ask at any time for an elaboration of the intent of a question, or for the geological rationale for including a question in the model, or for an ongoing trace of the effects of his answers on Prospector's conclusions. The intent is to provide the user with many of the services that could be provided by giving him telephone access to a panel of senior economic geologists, each an authority on a particular class of ore deposits.

The performance of Prospector depends on the number of models it contains, the types of deposits modeled, and the quality and completeness of each model. Because the Prospector program is primarily a research project, its coverage is still incomplete. It currently contains five prospect-scale models, one regional-scale model, and one drilling site selection model. The prospect-scale models consist of a Kuroko-type massive sulfide model contributed by Charles F. Park, Jr., a Mississippi-Valley-type carbonate lead/zinc model contributed by Neil Campbell, a near-continental-margin porphyry copper model contributed by Marco T. Einaudi, a Komatiitic nickel sulfide model contributed by Anthony J. Naldrett, and a Western-states sandstone uranium model contributed by Ruffin I. Rackley. The regional scale model is a variation of Mr. Rackley's model. The drilling site selection model, for porphyry copper deposits, was contributed by Mr. Victor Hollister, and differs somewhat from the other models: it derives its inputs from digitized maps of geological characteristics, and produces as output a color-coded graphical display of the favorability of each cell on a grid corresponding to the input map. These models were selected for a variety of reasons, including their economic significance, the extent to which they are well understood scientifically, the availability of expert geologists who could collaborate with us in the model development, and the new research issues that their implementation would raise.

Each model is encoded as a separate data structure, independent of the Prospector system per se. Thus, the Prospector program should not be confused with its models. Rather, Prospector should be thought of as a general mechanism for delivering relevant expert information about ore deposits to a user who can supply it with data about a particular prospect or region.

This paper describes briefly the process of developing and encoding such models for Prospector. General overviews of the technical principles are given in Hart, Duda and Einaudi (1978), mathematical aspects in Duda et al. (1976, 1978a), and detailed expositions in Duda et al. (1977, 1978b) and Hart, Duda and Konolige (1978).

III. FORMALISM FOR ENCODING EXPLORATION MODELS

A. Inference Networks of Assertions

For use in Prospector an ore deposit model must be encoded as a so-called inference network, a network of connections or relations between field evidence and important geological hypotheses. Since we sometimes do not wish to distinguish between evidence and hypotheses, we shall refer to either one as an assertion. To illustrate these ideas, we shall draw upon examples taken from M.T. Einaudi's porphyry copper model, which we shall denote by PCDA. Typical assertions in PCDA are "Hornblende has been pervasively altered to biotite" and "The alteration suggests the potassic zone of a porphyry copper deposit." The former would normally be thought of as field evidence, the latter as a geological hypothesis. A small portion of the PCDA inference network is shown in Figure 1. Here the terminal or "leaf" nodes correspond to field evidence asked of the user, while the other nodes represent hypotheses. The text in the boxes in Figure 1 is concise for reasons of graphical display; the actual questions asked of the user are more definitive.

Although assertions are statement that should be either true or false, in a given situation there is usually uncertainty as to whether they are true or false. Initially, the state of each assertion is simply unknown. As evidence is gathered, some assertions may be definitely established, whereas others may become only more

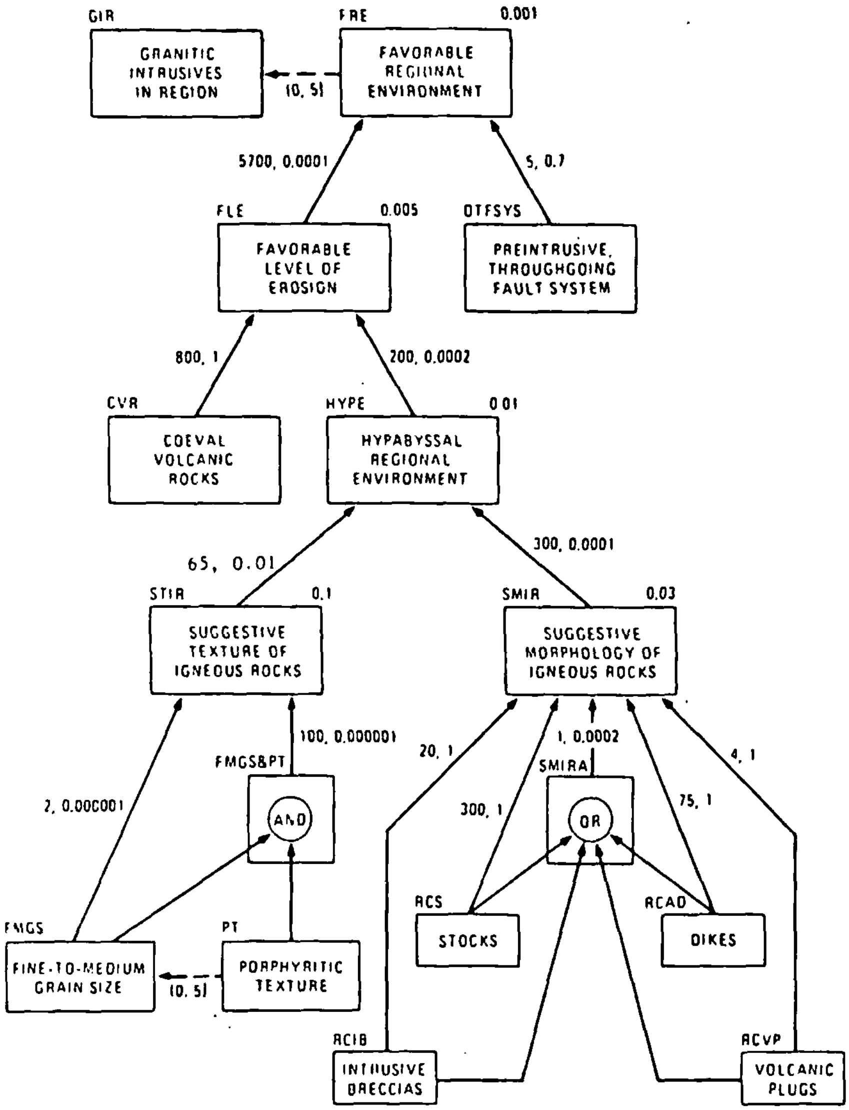


Figure 1. Portion of a Prospector model for porphyry copper deposits. See text for interpretation

REGIONAL ENVIRONMENT

or less likely. In general, we associate a probability value with every assertion. The "connections" in the inference network determine how a change in the probability of one assertion will affect those of other assertions.

The principal or top-level assertion in an inference network for a model is the assertion that the available evidence matches that particular model. To establish this assertion, it is usually necessary to establish several major factors. For example, to establish the top-level assertion in PCDA, we must establish the following hypotheses:

1. The petrotectonic setting is favorable for PCDA;
2. The regional evironment is favorable for PCDA;
3. There is an intrusive system that is favorable for PCDA.

Were any of these assertions field-observable evidence, they could be established merely by asking the user of the program whether they were true. However, since all of these factors are hypotheses, each must be further related to other factors. For example, the favorability of the petrotectonic setting can be established through the following three factors, each of which happens to be determinable (at least in principle) from observational evidence:

1. The prospect lies in a continental margin mobile belt;
2. The age of the belt is post-Paleozoic;
3. The prospect is subject to tectonic and magmatic activity related to subduction.

In general, the ore deposit models in Prospector have this type of hierarchical structure. The top-level assertion is determined by several major second-level assertions, each of which may be determined by third-level assertions, with this refinement continuing until assertions are reached that can be established directly from field evidence. This is illustrated in Figure 1, which shows graphically that portion of the PCDA model that describes the regional environment. In addition to this "top-to-bottom" development in terms of successive levels of assertions, the models also often exhibit a "left-to-right" organization in terms of spatial scale, from the petrotectonic setting on the left to the local details of mineralization and texture on the right. Exactly how these considerations interact is determined by the relations that exist among the assertions. The following section explains the nature of these relations and illustrates their occurrence in Figure 1.

B. Relations

Three basically different kinds of relations are used in Prospector to specify how a change in the probability of one assertion affects the probability of other assertions. We distinguish these as logical relations, plausible relations, and contextual relations.

1. *Logical Relations*. With logical relations, the truth (or falsity) of a hypothesis is completely determined by the truth (or falsity) of the assertions that define it. Such relations are composed out of the primitive operations of conjunction (AND), disjunction (OR), and negation (NOT). When several assertions must all be true for a hypothesis to be true, the hypothesis is the conjunction of the assertions. When the hypothesis is true if any of the assertions is true, the hypothesis is the disjunction of the assertions. Negation merely complements an assertion, interchanging truth and falsity. As an example of a logical relation, the PCDA model says that alteration of plagioclase is indicative of the barren-core zone if

1. Plagioclase has been altered to
   a. albite
   or
   b. minor sericite (or both)

and

2. Plagioclase has not been altered to major epidote.

Other examples of logical relations are shown in Figure 1.

Of course, in general we do not know whether the assertions are true, but can only estimate a probability or degree of belief that they are true. With logical relations, to compute the probability of a hypothesis from the probability of its component assertions we employ the fuzzy-set formulas of Zadeh (1965). Using these formulas, the probability of a hypothesis that is defined as the logical conjunction (AND) of several pieces of evidence equals the minimum of the probability values corresponding to the evidence. Similarly, a hypothesis defined as the logical disjunction (OR) of its evidence spaces is assigned a probability value equal to the maximum of those values assigned to the evidence spaces. One property of this procedure is that it often gives no "partial credit." In particular, if all but one of the assertions have been established, but the user can not even guess about the last, then the probability of their conjunction often remains at the value it had when the states of none of the assertions were known. This may be the appropriate conclusion. When it is not, one has the option of using plausible relations.

2. *Plausible Relations.* With plausible relations, each assertion contributes "votes" for or against the truth of the hypothesis. This would be expressed by relating the assertions to the hypothesis through a set of plausible inference rules. Each rule has an associated rule strength that measures the degree to which a change in the probability of the evidence assertion changes the probability of the hypothesis. This change can be positive or negative, since as assertion can be either favorable or unfavorable for a hypothesis. As with all parts of a model, these rule strengths are obtained by interviewing an authority on the corresponding class of ore deposits. Initially he may express the strengths in verbal terms, such as "rather discouraging" or "very encouraging." This is ultimately translated into numerical terms (as shown in Figure 1), the changes in probability being computed in accordance with the rules of Bayesian probability theory, as outlined below and described in detail in Duda et al. (1977).

Prospector's plausible reasoning scheme is based on Bayesian decision theory (Raiffa, 1968), exploiting an elementary theorem of probability known as Bayes' rule. For our purposes, the so-called "odds-likelihood" form of the rule is most convenient. This form relates three quantities involving an evidence assertion E and a hypothesis assertion H: the prior odds O(H) on the hypothesis, the posterior odds O(H | E) on the hypothesis, given that E is observed to be present, and a measure of sufficiency LS. Then Bayes' rule can be stated as

$$O(H \mid E) = LS * O(H) \quad (1)$$

Odds and probabilities are freely interchangeable through the simple relation $O = P / (1 - P)$, where P denotes probability, and hence $P = O / (1 + O)$. The sufficiency measure LS is a standard quantity in statistics called the likelihood ratio, and is defined by

$$LS = \frac{P(E|H)}{P(E|\sim H)} \tag{2}$$

where ~H means "not H."

Equation (1) prescribes a means for updating the probability (or odds) on H, given that the evidence E is observed to be present. An inference rule for which LS is large means that the observation of E is encouraging for H – in the extreme case of LS approaching infinity, E is sufficient to establish H in a strict logical sense. On the other hand, if LS is much less than unity, then the observation of E is discouraging for H, inasmuch as the observation of E diminishes the odds on H.

A complementary set of equations describes the case in which E is known to be absent, that is, when ~E is true. In this case, we can use Bayes' rule to write

$$O(H \mid \sim E) = LN * O(H) \tag{3}$$

where

$$LN = \frac{P(\sim E \mid H)}{P(\sim E \mid \sim H)} \tag{4}$$

The quantity LN is called the necessity measure. If LN is much less than unity, the known absence of E transforms neutral prior odds on H into very small posterior odds in favor of H. In the extreme case of LN approaching zero, E is logically necessary for H. On the other hand, if LN is large, then the absence of E is encouraging for H.

Hence to define an inference rule

IF E

THEN (to degree LS, LN) H,

the model designer must articulate E and H, and must supply numerical values for LS, LN, and O(H).

In general, the user may not be able to state that E is either definitely present or definitely absent. In this case, the updating formulas (1) and (3) cannot be applied directly, but can be extended to accommodate the uncertainty in the evidence. The extension used in Prospector involves a linear interpolation between the extremes of E's being definitely present or definitely absent. See Duda (1976, 1977) for details. The user expresses his certainty about E on an arbitrary –5 to 5 scale, where 5 denotes that the evidence is definitely present, –5 denotes that it is definitely absent, 0 indicates no information, and intermediate values denote degrees of certainty.

We illustrate this plausible inference scheme with examples taken from Figure 1. The two numbers associated with each inference rule in Figure 1 are its LS and LN values, respectively. The number appearing above each box representing a nonterminal node is the prior probability of that assertion's being true. For example, the figure indicates that the existence of stocks is a more highly sufficient indicator of "suggestive morphology of igneous rocks" (i.e., LS = 300) than is the existence of either dikes, intrusive breccias, or volcanic plugs (i.e., LS = 75, 20, and 4, respectively). Similarly, "favorable level of erosion" (FLE) is a highly sufficient and highly necessary factor for establishing "favorable regional environment" (i.e., LS = 5700 and LN = 0.0001), whereas the existence of a "preintrusive throughgoing fault system" (OTFSYS) is only mildly sufficient and mildly necessary for establishing "favorable regional environment." Hence the positive (LS) or negative (LN) votes of FLE are weighted much more heavily than those of OTFSYS.

The section of the model concerned with establishing "suggestive morphology of igneous rocks" (SMIR) illustrates how logical and plausible relations can be combined as building blocks to accomplish the intent of the economic geologist designing the model. This section of the PCDA model can be described as follows. "There are four positive indicators for establishing a suggestive morphology for igneous rocks (SMIR), namely intrusive breccias, stocks, dikes, and volcanic plugs. Each of these factors contributes independently to establishing SMIR, although to differing degrees. The absence of any one of these four factors individually is unimportant [i.e., LN = 1 for those rules]. However, if it is known that none of these factors is present [implying that the disjunction node SMIRA is false], then the probability of a suggestive morphology of igneous rocks is essentially zero [LN = 0.0002 for SMIRA]." In defining an inference network for a model, the object is to induce the model designer to articulate such statements, and then to translate the statements into network constructions.

To see how the effect of a piece of evidence propagates upward through the model, suppose that the user has indicated only that intrusive breccias are present, but this is definite. This fact multiplies the odds of SMIR by a factor of 20, hence raising its probability from 0.03 to 0.382. (The prior odds on SMIR are 0.03 / (1 – 0.03) = 0.030927, giving posterior odds on SMIR equal to 20 * 0.030927 = .61855, which corresponds to a probability of 0.61855 / (1+0.61855) = 0.382.) This in turn increase the odds on HYPE by a factor of 300 weighted by the degree to which SMIR has increased from its prior probability, i.e, by the factor 300 * (0.382–0.03) / (1–0.03) = 108.866. Hence the posterior probability of HYPE is 0.52373, which in turn increases the odds of FLE by a factor of 200 * (0.52373–0.01) / (1–0.01) = 103.78, giving a posterior probability for FLE of 0.34276. The propagation continues in this manner upward through the network.

It should be noted that Prospector expresses its conclusions to the user on the same -5 to 5 certainty scale that the user employs to express his certainty about evidence requested by the system. Prospector maps internal probability values to external certainty scores in a piecewise linear fashion, such that the posterior certainty is proportional to the difference between the posterior probability and the prior probability. For example, since the prior probability of

FLE is 0.005, a posterior probability of 0.34276 corresponds to a posterior certainty of 5 * (0.34276−0.005) / (1−0.005) = 1.697. Similarly, a posterior probability of 0.001 corresponds in this case to a posterior certainty of 5 * (0.001 − 0.005) / 0.005 = −4. See Shortliffe (1975) for a description of the subjective certainty scale used in the MYCIN medical diagnosis system.

3. *Contextual Relations.* It sometimes happens that assertions cannot be considered in an arbitrary order, but must be considered in a particular sequence. For example, one should determine that there is a relevant continental margin mobile belt before considering its age. This is more than a matter of preference, since it would be meaningless for the program to ask about the age of a non-existent belt.

To treat such situations we employ the third class of relations, contextual relations. In general, we use contexts to express a condition that must be established before an assertion can be used in the reasoning process. In the above example, the existence of a continental margin mobile belt would be specified as a context for asking about the age of the belt. Thus, before inquiring about the age, the system would employ all its resources to establish the existence of the belt, and would not ask about its age unless the probability of the belt were greater than its initial value.

Contextual relations are also used when one assertion is geologically significant only if another assertion has already been established. In such instances it would not be nonsensical to ask the former question without first establishing the latter, but it is the case that the former evidence is geologically irrelevant without the latter to establishing a match to the model. Two such instances are depicted by dashed arrows in Figure 1. In one of these instances, the entire "favorable regional environment" section of PCDA model will not be pursued unless it has first been determined that there are granitic intrusives in the region.

IV. OVERVIEW OF THE MODEL DEVELOPMENT PROCESS

Although the development and encoding of a model for Prospector is not a routine process, it does progress through several distinct phases whose general nature can be described. The four most important phases are summarized below.

A. Initial Preparation

Model development is a cooperative enterprise involving an exploration geologist, who is an authority on the type of deposit being modeled, and a computer scientist who understands the operation of the Prospector system. The first step in developing a model is one of introducing the exploration geologist (model designer) to the inference network formalism, and introducing the computer scientist (model implementor) to the general nature of the class of deposits being modeled. In particular, this includes the identification of several known deposits that should fit the model well, and several known deposits that may fit partially, but that lack certain important characteristics. These specific cases help to establish the various factors that must be taken into account.

B. Initial Design

The initial design of the inference network is the most creative phase of the process. It requires the identification of the various assertions, the organization of the assertions into a hierarchical structure (as illustrated in Figure 1), the determination of the types of relations (logical, plausible, and contextual) that exist among assertions, and the estimation of values for the parameters (the voting strengths and initial probabilities). The magnitude of this task depends upon the size and complexity of the model being developed; as a point of reference, the smallest model currently in Prospector contains 28 assertions and 20 inference rules, while the largest contains 212 assertions and 133 inference rules. The initial design is usually facilitated by considering factors in the "top-down" and "left-to-right" sequence described earlier. Delicate refinement is best avoided at this time, since subsequent revision often causes significant sections of the model to be reorganized, enlarged, or otherwise modified.

In addition to the connections between assertions exhibited directly by the inference network, there are connections that exist because of the geological meaning of the assertions. For example, the statement that there are sulfide minerals is obviously related to the statement that there is pyrite in quartz veins; assertion of the latter implies the former, and denial of the former denies the latter. Recognition of such connections within a model avoids redundant or foolish questioning; recognition of such connections between different models allows the program to consider more than one deposit class at a time. Prospector can automatically recognize many of these assertions if each assertion is properly articulated. This articulation, which is described in more detail in Duda (1978b), should also be completed during the initial design.

C. Installation and Debugging of the Model

At the end of Phase B, the model exists in a "pencil-and-paper" form. To be incorporated into the program, the encoding must be given a formal description. This is done through the use of a model description language (see Duda, 1978b). The details of this language are not particularly important here. However, the task itself is important; upon its completion the program can be run, and accidental blunders or bugs can be corrected. In addition, the program can produce a questionnaire for the model that is useful in gathering data for subsequent testing and revision.

D. Performance Evaluation and Model Revision

Given the questionnaire data for a number of actual deposits, it is possible to make a serious quantitative evaluation of how well particular deposits match the model. In our experience, this evaluation inevitably exposes various shortcomings of the model as encoded, requiring revision of the work done in Phases B and C. Some care must be exercised here to avoid "overfitting" the model to the data. In general, the goal is to produce a model that can discriminate different types of deposits without losing the ability to generalize, so as to allow for the variations one would expect in new situations. Achievement of that goal currently remains as much an art as a science. The following section describes in some detail the use of simple performance evaluation techniques as an aid to refining a model.

V. USE OF PERFORMANCE EVALUATION IN REFINING A MODEL

To demonstrate that the performance of an expert knowledge-based system is (or is not) comparable to that of the experts it emulates, it is useful to subject the system to an appropriate objective evaluation. The simple performance evaluation experiments reported in this section serve several purposes: (1) to provide an objective, detailed, quantitative measure of the current performance of a model; (2) to pinpoint those sections of the model that are not performing exactly as intended, thereby establishing priorities for future revisions; (3) to assess consistency of performance across different exploration sites.

We now evaluate a model for a class of porphyry copper deposits (PCDA) designed by Prof. Marco Einaudi of Stanford University. Input data were available for three test cases, namely, the known deposits called Yerington (Nevada), Bingham (Utah), and Kalamazoo (Arizona), each of which is considered an exemplar of the PCDA model.[2] On the −5 to 5 certainty scale described earlier, the overall certainty scores computed by Prospector are 4.769 for the Yerington deposit, 4.721 for Bingham, and 4.756 for Kalamazoo, indicating a good match of these sites to the PCDA model.

To show performance in detail, we give below the hierarchical structure of the major sections of the PCDA model. Included at the right in this enumeration is the total number of questions that may be asked by Prospector for each of the major sections of the model, thus showing the relative distribution of these questions. (The questions in the FAMR section may be asked several times during a consultation session, once for each geographically distinct zone within the prospect area. Each such zone has relatively homogeneous geological characteristics, as determined by the user.)

	Total Number of Questions Defined in PCDA Model (Version 2)
Porphyry Copper deposit, type A (PCDA)	81
Favorable petrotectonic setting (FPTS)	4
Favorable regional environment (FRE)	9
Favorable PCDA intrusive system (FPCDAIS)	68
Favorable composition in differentiated sequence (FCDS)	4
Favorable intrusive system (FIS)	9
Favorable alteration and mineralization relations (FAMR)	56

[2] The questionnaire input data used in the present tests are reported in Duda et al., 1978b, pp.185-93.

As a calibration exercise, Prof. Einaudi offered a target value for the certainty score that should be assigned to each of the three deposits for each of the major components of the model listed above, based on the values in the input data set for each prospect site. The target values are given either in the form of a single number (on a –5 to 5 scale), or as two numbers establishing an upper and lower bound on a certainty interval. The estimates are listed in Table 1 on the left for each site in turn, with the scores as determined by execution of Prospector recorded on the right. (We informed Prof. Einaudi of the values on the right only after he had given us those on the left.)

Yerington Deposit

Name of Model Node	Einaudi's Estimate	Prospector Score
PCDA	4.5 to 5.0	4.769
FPTS	4.5 to 5.0	4.528
FRE	4.5	4.540
FPCDAIS	4.5 to 5.0	4.787
FCDS	5	4.524
FIS	5	4.744
FAMR	4.5 to 5.0	4.225

Bingham Deposit

Name of Model Node	Einaudi's Estimate	Prospector Score
PCDA	4.5	4.721
FPTS	3.5 to 4.0	4.449
FRE	4.0 to 4.5	4.829
FPCDAIS	4.5 to 5.0	4.729
FCDS	5	2.407
FIS	5	4.744
FAMR	4.0	4.225

Kalamazoo Deposit

Name of Model Node	Einaudi's Estimate	Prospector Score
PCDA	4.0 to 4.5	4.756
FPTS	4.0 to 4.5	4.449
FRE	3.5	1.784
FPCDAIS	4.5 to 5.0	4.791
FCDS	5	4.722
FIS	5	4.744
FAMR	4.0	4.225

Table 1. Prospector Scores for Several Levels of the PCDA Model (Version 2)

The data in Table 1 show that Prospector scores each of these sections of the model with high certainty for each site, with the exception that model node FCDS for the Bingham deposit and model node FRE for the Kalamazoo deposit are scored somewhat lower. In most cases shown in Table 1 Prospector agrees very closely with Prof. Einaudi's estimate. These conclusions can be expressed quantitatively by first identifying the values in Table 1 with a concise notation, then defining a simple formula for the relative error of Prospector in predicting Prof. Einaudi's estimates. Thus:

Let C(X, Y, Z) = Certainty score given to model node Z by agent X for site Y,
where X denotes either Prospector or Einaudi

For example, C(Prospector, Yerington, FPCDAIS) = 4.787. When Einaudi gave an interval of certainty values instead of a single value, we use the mid-point of the interval as the value of C. Then an error measure is given by

$$E(Y, Z) = \frac{C(\text{Einaudi}, Y, Z) - C(\text{Prospector}, Y, Z)}{C(\text{Einaudi}, Y, Z)}$$

For example, E(Yerington, FPCDAIS) = (4.75 − 4.787) / 4.75 = −0.008, meaning that Prospector's prediction is accurate to within 0.8% in this case. Since Table 1 gives values for seven nodes of the model for each of three known deposits, we can compute the value of E for 21 different instances. For 5 of the 21 data points Prospector predicted Einaudi's estimate to within 1%, while 15 of the 21 data points show agreement to within 10%. The grand average over the 21 data points is 10.3%. For convenience, we list these 21 values of E in Table 2, expressed as percentages.

	Yerington	Bingham	Kalamazoo	Average of Absolute Values
PCDA	−.3 %	−4.9 %	−11.9 %	5.7 %
FPTS	4.7	−18.6	−4.7	9.3
FRE	−.9	−13.6	49.0	21.2
FPCDAIS	−.8	.4	−.9	.7
FCDS	9.5	51.9	5.6	22.3
FIS	5.1	5.1	5.1	5.1
FAMR	11.1	5.6	5.6	7.6
Average of Absolute values:	4.1	14.3	11.8	10.3

Table 2. Relative Error (E) of Prospector Scores as Predictors of Einaudi's Estimates (derived from data in Table 1)

Inspection of Table 2 indicates that efforts to revise the PCDA model should focus on the FRE and FCDS sections. When such revisions are completed, an updated version of Table 2 will indicate the extent to which the revisions achieved the objectives that motivated them.

The small number of cases tested, and the fact that all the present test cases are exemplars of the PCDA model, and the fact that the model designer himself supplied the input data concerning the test cases, are limitations; the present tests are more necessary than sufficient conditions for good performance. Despite these limitations, the preliminary results reported here have proved useful in the ongoing model refinement process. More extensive performance evaluation results are reported in Duda et al. (1978b).

VI. REMARKS

This paper has outlined the typical procedures used to develop an exploration model for the Prospector system. We have described the inference network and Bayesian propagation scheme underlying Prospector models, and we have illustrated the use of simple performance evaluation techniques in "fine-tuning" a model systematically. Our experience indicates that the model design process inherently requires feedback. Although different problem solving domains differ in many details, we believe the process of constructing Prospector-like plausible reasoning systems follows certain general patterns and stages of development such as are described here. Hence we have presented a concrete case study, in the domain of mineral exploration, that may credibly suggest what might be expected in attempts to apply a similar methodology to other domains of expertise.

Besides the running program, there appear to be several other benefits to this type of expert system approach. The model design process challenges the model designer to articulate, organize, and quantify his expertise. Without exception, the economic geologists who have designed Prospector models have reported that the experience aided and sharpened their own thinking on the subject matter of the model. In addition, most of the geologists we know who have had experience with Prospector have remarked about its potential value as an educational tool. In this regard, the models in the system contain explicit, detailed information synthesized from the literature and the experience of expert explorationists, together with explanatory text that can be obtained upon request. Furthermore, a typical consultation session with Prospector costs only about $10 at current commercial computer rates.

VII. ACKNOWLEDGEMENTS

Prospector would not exist without the talents and efforts contributed by many economic geologists and computer scientists. Among these, we are particularly indebted to the economic geologists Alan N. Campbell, Neil Campbell, Marco T. Einaudi, Victor F. Hollister, Anthony J. Naldrett, Charles F. Park, Jr., and Ruffin I.Rackley, and to the computer scientists Phyllis Barrett, Kurt Konolige, Nils J.Nilsson, Rene Reboh, Jonathan Slocum, and B.Michael Wilber.

REFERENCES

Duda, R.O., P.E. Hart and N.J. Nilsson (1976), 'Subjective Bayesian methods for rule-based inference systems,' *Proc. National Computer Conference,* (AFIPS Conference Proceedings, Vol. 45), pp. 1075-1082, 1976.

Duda, R.O., P.E. Hart, N.J. Nilsson, R. Reboh, J. Slocum and G.L. Sutherland (1977), 'Development of a computer-based consultant for mineral exploration,' Annual Report, SRI Projects 5821 and 6415, SRI International, Menlo Park, California, October 1977.

Duda, R.O., P.E. Hart, N.J. Nilsson, and G.L. Sutherland (1978a), 'Semantic network representations in rule-based inference systems,' in *Pattern Directed Inference Systems,* D.A. Waterman and F. Hayes-Roth (Eds.), pp. 203-221, Academic Press, New York, 1978.

Duda, R.O., P.E.Hart, P.Barrett, J.Gaschnig, K.Konolige, R.Reboh and J.Slocum (1978b), "Development of the Prospector system for mineral exploration," Final Report, SRI Projects 5821 and 6415, SRI International, Menlo Park, California, October 1978.

Fagan, L.M. (1978), "Ventilator Manager: A program to provide on-line consultative advice in the intensive-care unit." Heuristic Programming Project Memo HPP-78-16, Dept. of Computer Science, Stanford University, September 1978.

Hart, P.E., R.O. Duda, and K. Konolige (1978), "A computer-based consultant for mineral exploration," Second Semiannual Report, SRI Project 6415, SRI International, Menlo Park, California, April 1978.

Hart, P.E., R.O. Duda, and M.T. Einaudi (1978), "A computer-based consultation system for mineral exploration," to appear in *Computer Methods for the 80's,* Society of Mining Engineers of the AIME, 1978.

Martin, N., P. Friedland, J. King, and M. Stefik (1977), "Knowledge Base Management for Experiment Planning in Molecular Genetics," *Proc. of 5th International Joint Conference on Artificial Intelligence,* pp. 882-887, Cambridge, Mass., August 1977.

Pople, H.E., Jr., J.D. Myers, and R.A. Miller (1975), "DIALOG: A model of diagnostic logic for internal medicine," *Proc. of 4th International Joint Conference on Artificial Intelligence.* pp. 848-855, Tbilisi, Georgia, USSR, 1975.

Raiffa, H. (1968), *Decision Analysis,* Addison-Wesley Publishing Co., New York, 1968.

Shortliffe, E.H. (1976), *Computer-Based Medical Consultations: MYCIN,* American Elsevier Publishing Co., New York, 1976.

Shortliffe, E.H., and B.G. Buchanan (1975), "A model of inexact reasoning in medicine," *Math. Biosci.,* Vol. 23, pp. 351-379, 1975.

Szolovits, P., and S.G. Pauker (1976), "Research on a medical consultation system for taking the present illness," *Proc. of 3rd Illinois Conference on Medical Information Systems,* pp. 299-320, University of Illinois at Chicago Circle, November 1976.

Weiss, S.M., C.A. Kulikowski, and A. Safir (1977), "A model-based consultation system for the long-term management of glaucoma," *Proc. of 5th International Joint Conference on Artificial Intelligence,* pp. 826-833, MIT, Cambridge, Massachusetts, 1977.

Yu, V.L., et al. (1978), "Evaluating the performance of a computer-based consultant," Heuristic Programming Project Memo HPP-78-17, Dept. of Computer Science, Stanford University, September 1978.

Zadeh, L.A. (1965), "Fuzzy sets," *Information and Control,* Vol. 8, pp. 338-353, 1965.

EG&G Idaho, Inc.

P.O. BOX 1625, IDAHO FALLS, IDAHO 83415

September 6, 1984

Dr. Mike Wright
UURI/ESL
391 Chipeta Way, Suite C
Salt Lake City, UT 84108

EXPERT SYSTEMS READING MATERIAL - BWD-5-84

Dear Mike:

I enjoyed meeting you yesterday to discuss expert systems applications in geology. As promised, I'm sending you some additional reading material, along with a reading guide to help you budget your time. The enclosed material will augment the papers you already have in providing background information for your discussions with Marshall Reed.

Very truly yours,

Brent Dixon

Brent Dixon
Advanced Methods Branch

ts

—0717

Enclosures:
As Stated

EG&G Idaho, Inc.
P.O. BOX 1625, IDAHO FALLS, IDAHO 83415

October 15, 1984

Marshall Reed
Forrestal Building
1000 Independence Ave. S.W.
CE 324 5H081
Washington, D.C. 20585

GEOTHERMAL "EXPERT" SYSTEM WRITE-UP - RAM-64-84

Dear Mr. Reed:

Enclosed is a brief write-up describing the potential advantages for a geothermal "expert" system. We have been working for several years in house at EG&G developing a capability in expert systems and I believe geothermal is an ideal application for this concept. I have discussed the idea with Mike Wright, Sue Prestwich and Clay Nichols and they all agree geothermal is a good application.

Mike indicated he has talked with you about the concept and that you were interested. Some of our people with expertise in expert systems will be in Washington on November 14th and 15th. If you are interested, they would be glad to stop by your office and give you a more complete presentation on the background of expert systems and our ideas for their use in geothermal systems.

If you have any questions, give me a call at 526-9688.

Sincerely,

J H Ramsthaler

J. H. Ramsthaler
Hydropower/Geothermal Programs

ks

Enclosure:
As Stated

S. M. Prestwich, DOE-ID
M. Wright, UURI
J. O. Zane, EG&G Idaho (w/o Enc.)

GEOTHERMAL EXPERT SYSTEM

The Problem:

Geothermal exploration and site development require integration of expertise from the fields of geology, geochemistry, geophysics, and reservoir engineering. Typically, a team of experts from each field are brought together to interpret the data. The results can vary, depending on the true level of expertise of each member, biases of the team leader toward a particular field, "political" biases, and variances in human performance versus potential. Even with the same data, two teams (or one team on two occasions) will derive different conclusions on million dollar decisions. A case in point is the Bacca geothermal project in New Mexico: Using the same data the exploration team was 8 for 8 on successful wells while the development team was 0 for 8.

Expert Systems:

Expert systems are an application of artificial intelligence research in which computer programs are designed to mimic the problem solving methods of human experts. Human experts tend to solve problems by using "hunches" and "rules of thumb" developed from their many years of experience. Expert systems use this same approach, using logic and heuristics to make educated guesses when necessary, to recognize promising approaches to problems, and to deal effectively with erroneous or incomplete data. Expert systems consist of a "knowledge base" containing thumb rules, along with control and user interface programs. The rules are "taught" to the system by human experts, much like they would groom an apprentice. For example, a geothermal rule might look like this:

IF The target area contains several thermal springs, and
The spring discharge temperatures vary, and
The spring discharge rates vary, and
Geothermometers indicate an at-depth temperature x, and
The spring geochemistry correlation with temperature x is high,

THEN Increase the confidence level of the geothermometers.

The actual building process consists of repeated teaching and testing.

Expert systems are capable of documenting their reasoning process, both from the aspect of how a piece of data will be used and, later, how a solution was obtained. They are consistent, thorough, and won't become susceptible to biases or boredom. Because of the modularity of their programs (the rules), they can easily be changed to reflect changes in the state-of-the-art in their field.

Geologic Expert Systems:

Geology is a good expert system problem domain because it is an ever-changing field with few governing equations, lots of uncertainty, high data acquisition costs, and a small number of truly expert people. A number of geologic expert systems have been built to date to help disseminate the knowledge of these rare experts, including:

PROSPECTOR - a mineral exploration expert system which uses surface geology and well logs to find ore bodies,

WAVES - an oil exploration expert system which interprets seismic data,

DIPMETER ADVISOR - a specialist in modeling strata from dipmeter logs,

DRILLING ADVISOR - a drilling trouble-shooter which diagnoses problems and recommends both preventative and corrective measures.

The Geothermal Expert System:

A geothermal expert system could be used to locate unknown fields, find drilling sites within known fields, aid in field development, or aid in the actual drilling. The actual problem to be targeted would depend on need and the availability of experts. Ideally, the system would be designed and built and then turned over to the private sector for additional training and "maintenance" (in a changing field such as geology a system must remain current to be of use). Required personnel would include experts in each required discipline, knowledge engineers (people who design expert system internals and aid in their training), and support staff. A proposed implementation plan is as follows:

Phase 1 -- Select target problem (field location, site selection, field development, drilling aid, etc.)
Build and assess prototype for selected problem
1 year, $200K

Phase 2 -- Train and test system -
2 years, $400K/yr

Phase 3 -- Final system polishing -
Technology transfer
1 year, $400K

Phase 4 -- Transfer complete, consultation support -
1 year, $100K

FY-85 Geothermal Expert System - Phase 1

Target Problem Selection 20K

A target problem within the geothermal arena will be chosen based on discussions with field experts, potential users, and the program management. The selected problem must be addressable by expert system technology, experts must be available, and the solution of the problem must be non-trivial and useful to the potential end user.

Expert Team Selection 15K

Individual personnel with acknowledged expertise in the required fields will be contacted and a system training team formed. The team will meet to draw up the initial scope of the system and the basic knowledge structure will be mapped out.

Expert System Software Selection 25K

The system internal requirements will be developed by the knowledge engineers based on the problem scope and knowledge structure. The requirements will be compared against the capabilities of commercially available expert system building tools. Either a tool will be acquired, or in-house tool development will begin.

Prototype Development 125K

The expert team will develop rules and a seed knowledge base will be built. A set of test problems will be developed and the expert team will begin the test and train loop. The knowledge engineer will provide guidance in rule composition, hybridize the building tool as required, and search the knowledge base for holes and inconsistencies.

Prototype Demonstration and Assessment 25K

Toward the end of the year development will be stopped and a demonstration of the system held for program management. At this time the system will be assessed and the development effort redirected as needed.

EXPERT SYSTEMS

DEFINITION: A COMPUTER PROGRAM WHICH USES EXPERT KNOWLEDGE AND INFERENCE PROCEDURES TO SOLVE LOGIC PROBLEMS.

- CAN ACT AS INTERMEDIARY BETWEEN AN EXPERT AND A USER OF THE EXPERTISE TO MAKE THE EXPERTISE MORE WIDELY AVAILABLE.

EXAMPLES:

MEDICAL DIAGNOSIS
DATA ANALYSIS (EG. REACTOR ACCIDENT DIAGNOSIS)
DESIGN PROBLEMS
PROSPECTING
LAW CONSULTANT

ARTIFICIAL INTELLIGENCE VS. NATURAL INTELLIGENCE

AI	NATURAL
Permanence	Perishable
Ease of Duplication	Apprenticeship
Less Expensive	Can Be Costly
Consistent and Thorough	Erratic
Documentable	Difficult to Reproduce or Recall
Uninspired	Creative
Tailored Knowledge	Learns
Symbolic Input	Uses Sensory Data As Well As Symbols
Narrow Focus	Use Wide Context of Knowledge (Common Sense)

EXPERT SYSTEMS FOR RESOURCE EXPLORATION

• PROSPECTOR:	AN EXPERT SYSTEM THAT EVALUATES SITES FOR POTENTIAL MINERAL DEPOSITS
• DIPMETER ADVISOR:	AN EXPERT SYSTEM THAT ANALYZES INFORMATION FROM OIL WELL LOGS
• WAVES:	AN EXPERT SYSTEM THAT ADVISES ENGINEERS ON THE USE OF SEISMIC DATA ANALYSIS PROGRAMS FOR OIL INDUSTRY
• HYDRO:	A COMPUTER CONSULTATION SYSTEM FOR SOLVING WATER RESOURCE PROBLEMS
• DRILLING ADVISOR:	AN OPERATIONAL EXPERT SYSTEM FOR DIAGNOSING OIL WELL DRILLING PROBLEMS AND RECOMMENDING CORRECTIVE AND PREVENTIVE MEASURES

PROSPECTOR
DEVELOPED AT SRI

- AIDS GEOLOGIST IN EVALUATING THE MINERAL POTENTIAL OF A SITE OR REGION
- ACCEPTS FIELD OBSERVATIONS, PROVIDES FINDINGS AND REQUESTS ADDITIONAL INFORMATION
- INTENDED TO PROVIDE SERVICES LIKE TELEPHONE ACCESS TO A PANEL OF SENIOR GEOLOGISTS, EACH AN AUTHORITY ON A PARTICULAR CLASS OF ORE DEPOSITS
- INCLUDES MODELS FOR PORPHYRY COPPER AND MOLYBDENUM DEPOSITS, SANDSTONE URANIUM DEPOSITS, KOMATIITIC NICKEL, SULFIDE DEPOSITS, MASSIVE SULFIDE DEPOSITS AND MISSISSIPPI VALLEY-TYPE CARBONATE LEAD/ZINC DEPOSITS

DIPMETER ADVISOR

DEVELOPED BY SCHLUMBERGER-DOLL RESEARCH

- EMULATES HUMAN EXPERT PERFORMANCE AT DIPMETER INTERPRETATION

- DIPMETER MEASURES CONDUCTIVITY OF ROCK AS IT GOES DOWN A DRILL HOLE--DATA USED TO CHARACTERIZE WELL

PROPOSED GEOTHERMAL EXPERT SYSTEM

- WOULD DRAW FROM MANY SOURCES OF KNOWLEDGE
  - GEOLOGISTS
  - GEOCHEMISTS
  - GEOPHYSICISTS
  - RESERVOIR ENGINEERS
- BEST CONFIGURATION CAN'T BE PREDICTED AHEAD OF TIME
- IMPLEMENTATION PLAN

  PHASE 1 - FEASIBILITY STUDY, BUILD PROTOTYPE -
  1 YEAR, $200K

  PHASE 2 - TRAIN AND TEST SYSTEM -
  2 YEARS, $400K/YR

  PHASE 3 - FINISHING WORK, TECHNOLOGY TRANSFER -
  1 YEAR, $400K

  PHASE 4 - TRANSFER COMPLETE, CONSULTATION SUPPORT -
  1 YEAR, $100K

R1 Revisited: Four Years in the Trenches

Judith Bachant
Intelligent Systems Technology Group
Digital Equipment Corporation
Hudson, Massachusetts 01749

John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

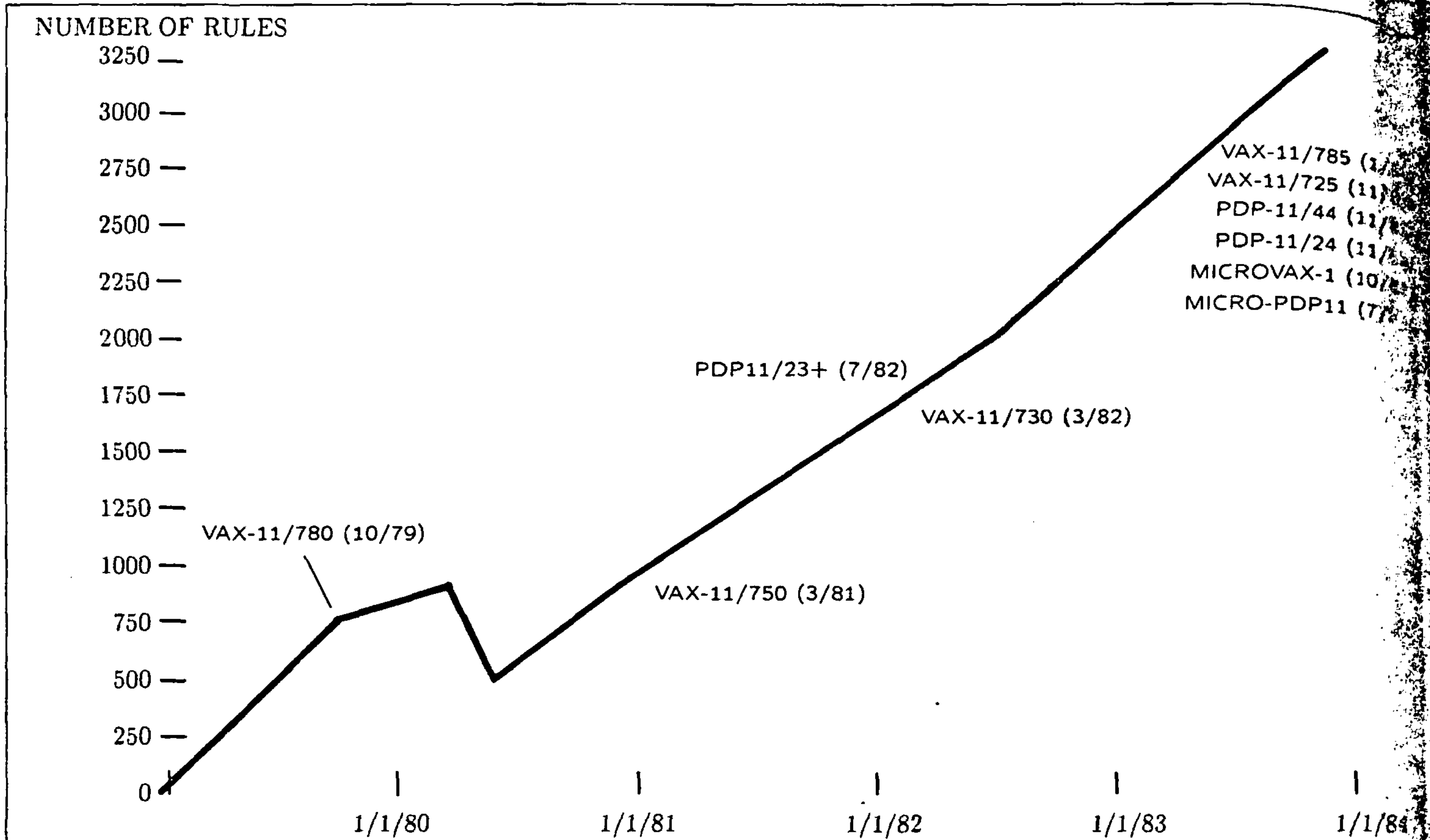
In 1980, Digital Equipment Corporation began to use a rule-based system called R1 by some and XCON by others to configure VAX-11 computer systems. In the intervening years, R1's knowledge has increased substantially and its usefulness to Digital continues to grow. This article describes what is involved in extending R1's knowledge base and evaluates R1's performance during the four year period.

IN THE SUMMER 1981 ISSUE of the AI Magazine, an article entitled "R1: the formative years" described how a rule-based configurer of computer systems had been developed and put to work (McDermott, 1981). At the time that article was written, R1 had been used for only a little over a year and no one had much perspective on its use or usefulness. R1 has now been configuring computer systems for over four years. This experience has provided some insight into the ease and difficulty of continuing to grow an expert system in a production environment and into the kind of performance expectations it might be reasonable to have about a current generation rule-based system.

The approach R1 takes to the configuration task and the way its knowledge is represented have been described elsewhere (McDermott,1980) and (McDermott, 1982). Briefly, given a customer's purchase order, R1 determines what, if any, substitutions and additions have to be made to the order to make it consistent, complete, and produce a number of diagrams showing the spatial and logical relationships among the 50 to 150 components that typically constitute a system. The program has been used on a regular basis by Digital Equipment Corporation's manufacturing organization since January, 1980. R1 has sufficient knowledge of the configuration domain and of the peculiarities of the various configuration constraints that at each step in a configuration task it is usually able to recognize just what to do; thus it ordinarily does not need to backtrack when configuring a computer system.

At the beginning of R1's development, no clear expectations existed about how long it would take to collect enough knowledge to make R1 an expert. We did expect that at some point the rate at which R1 would acquire new knowledge would at least slow, if not stop. We even thought that R1 would be done eventually (that is, R1 would enter a maintenance mode of well-defined and minor additions, interspersed with occasional bug fixes.) It is difficult now to believe R1 will ever be done; we expect it to continue to grow and evolve for as long as there is a configuration task. It may be that if R1's domain were less volatile, R1 would not require perpetual development. But it is

A large number of people have played critical roles in R1's development. Among those who deserve special mention are John Barnwell, Dick Caruso, Ken Gilbert, Keith Jensen, Allan Kent, Dave Kiernan, Arnold Kraft, Dennis O'Connor, and Ed Orciuch. We want to thank Allen Newell, Dennis O'Connor, and Ed Orciuch for their helpful comments on earlier drafts of this article.



R1's Growth

Figure 1.

probably also true that if the domain were less volatile, the task would not require a knowledge-based system.

The early expectations about R1's performance were likewise vague, except just as R1 was beginning to be used, a Digital employee responsible for the configuration process predicted that for R1 to be useful, 90% to 95% of its configurations would have to be perfectly correct. This performance goal is interesting, not so much because R1 took three years to reach it, but because it turned out to be completely wrong. R1's task is just one small part of a process designed to ensure that high quality computer systems are built. Significant redundancy exists in the process precisely because historically no individual has both known enough about configuration and been able to pay close enough attention to each order to be entrusted with the total responsibility. R1 was able to provide significant assistance even when it knew relatively little because the people who used R1 did not demand more of it than of its human predecessors. The one definite performance expectation almost everyone had about R1 in its early days was that it would always configure the same set of components in the same way. It is obvious now and should have been obvious then that this expectation could have been satisfied only if R1 had been discouraged from becoming more expert.

These expectations about R1's developmental and performance histories introduce the two parts of the article. In the next section, the focus will be on the kind of involvement required to extend R1's knowledge base. The final section's focus will be on the kinds of erroneous behavior R1 has exhibited.

R1's Developmental History

This section provides a somewhat anecdotal trip through R1's past. Although it mentions the first year, when most of the activity was at Carnegie-Mellon University [CMU], the primary focus is on the four following years, after R1 began to be used at Digital. When CMU handed over the initial version of R1 to Digital in January 1980, Digital scrambled to put an organization in place that could continue its development. This organization, currently known as the Intelligent Systems Technologies group, began with only five individuals, none of whom had any background in AI. Over the past four years, the group has grown to 77 people responsible for eight different knowledge-based systems, one of which is R1. As R1 was developed, an attempt was made to effect a division of labor between those people responsible for representing R1's knowledge and those responsible for collecting and validating that knowledge. Of the initial technical people, one was an engineer who played the roles of both a domain expert and of an interface to other domain experts outside the group; the other three people took the knowledge collected by the engineer and formulated it so it was compatible with R1's other knowledge. When the organization was a little over two years old the technical group had grown to eight people, five of whom were responsible for encoding the knowledge collected and validated by the other three. The size of the R1 technical group is still about eight. Now, however, less of a distinction exists between the people responsible for knowledge encoding and those responsible for knowledge collection.

The Knowledge R1 Acquired

Over the past four years, the amount of effort devoted to adding knowledge to R1 has remained relatively constant at about four worker-years per year. And R1's knowledge has grown at a relatively constant rate, though the focus has shifted around. At times the task of eliminating inadequacies in R1's configuration knowledge has received the most attention: at other times, the group's energies have been directed primarily at broadening R1's abilities in various ways. Figure 1 shows the rate at which R1's knowledge has grown; the points in time at which R1 became able to configure new system types are marked. Figure 1 does not show the amount of product information to which R1 has access. This information, which is stored in a data base, is a critical part of the body of information needed to configure a computer system correctly. R1 retrieves the description of each component ordered before it begins configuring a system; while configuring the system, if it determines some piece of required functionality is missing, it searches the data base for components that will provide that functionality. R1 currently has access to almost 5500 component descriptions. We do not have good data on the rate at which the data base has grown, but what data we have suggest the growth rate is quite irregular.

In this article, R1's growth is measured in number of rules. The following values hint at the amount of knowledge an R1 rule contains. The average conditional part of one of R1's rules has 6.1 elements (the minimum number is 1 and the maximum 17). Each element is a pattern that can be instantiated by an object defined by as many as 150 attributes. On the average, a pattern will mention 4.7 of those attributes (the minimum is 1 and the maximum 11) and restrict the values which will satisfy the pattern in various ways. The tests are mostly simple binary functions that determine whether some value in the object has the specified relationship to some constant or to some other value in that or another object. The action part of an average rule has 2.9 elements (the minimum is 1 and the maximum 10). Each element either creates a new object or modifies or deletes an existing object. A rule can be applied when all of its condition elements are instantiated.[1]

[1] For additional information about the nature of R1's rules as well as those of other systems written in OPS5, see (Gupta, 1983).

Work on R1 began in December 1978. During the first four months, most of the effort was on developing an initial set of central capabilities. The initial version of R1 was implemented in OPS4, a general-purpose rule-based language (Forgy, 1979). By April, R1 had 250 rules. During the same period, a small amount of effort was devoted to generating descriptions of the most common components supported on the VAX-11/780. After this demonstration version of R1 had been developed, most of the effort during the next six months was divided between refining those initial capabilities and adding component descriptions to the data base; in October 1979, R1 had 750 rules and a data base consisting of 450 component descriptions. During the following six months, little development work was done on R1 either at Digital or CMU because the main focus was on defining a career path for R1 within Digital. But beginning in April 1980, three months were spent at CMU in rewriting the OPS4 version of R1 in OPS5 (Forgy, 1981). Given that the knowledge was already laid out in the OPS4 version, a variety of generalizations emerged and the resulting system, though more capable, had only 500 rules.

By the end of 1980, R1 had 850 rules, most of which were added by people at CMU to provide R1 with additional functionality; the primary focus at Digital during the second half of 1980 was on adding component descriptions to the data base and providing a group of people with the skills to take over the continued development of R1. Most of the work on R1 since early in 1981 has been done by people at Digital. By March 1981, the group at Digital had extended R1 so it could configure VAX-11/750 systems. During the remainder of 1981, most of the group's effort was focused on refining R1's knowledge of how to configure VAX-11/780 and VAX-11/750 systems. In 1982, the focus changed to extending R1 to cover more systems. While some effort was spent in improving R1's performance, substantial effort was spent in extending its scope. By March, a few months before the VAX-11/730 was announced, R1 was able to configure VAX-11/730 systems, and by July, R1 was able to configure PDP-11/23+ systems. At that point, R1's knowledge base consisted of about 2000 rules. The remainder of 1982 and the first few months of 1983 were devoted primarily to refining that knowledge. At that point, a concerted effort was made to extend R1's capabilities so it could configure all the systems sold by Digital in significant volume. When that task was finished in November 1983, R1 had about 3300 rules and its data base contained about 5500 component descriptions. While a significant amount of time will continue to be devoted to extending R1's capabilities to cover new systems as they are announced, effort will also be spent in continuing to deepen R1's expertise in the configuration domain.

As Digital has become more dependent on R1, it has become increasingly important that R1 be highly reliable. Thus substantial attention has been paid to the question of how to combine the demands of reliability with those of continuous development. Early on, little attention was paid to formalizing the developmental process; as problems were reported,

	NUMBER OF RULES	AVERAGE RULES PER SUBTASK	AVERAGE NUMBER OF PARTS ORDERED	AVERAGE RULE FIRINGS	PERCENT OF KNOWLEDGE FREQUENTLY USED	NUMBER OF PARTS IN THE DATABASE
THE INITIAL R1	777	7.6	88	1056	44%	420
THE CURRENT R1	3303	10.3	78	1064	47%	5481
VAX-11/785	2883	9.8	163	2654	24%	3398
VAX-11/780	2883	9.8	171	1925	31%	3398
VAX-11/750	2801	9.7	111	1300	29%	2915
VAX-11/730	2810	9.7	85	1141	29%	2489
VAX-11/725	2788	9.7	34	622	8%	1981
MICROVAX-1	1516	7.3	34	546	18%	1490
MICRO-PDP11	1516	7.3	44	546	18%	1828
PDP-11/23+	1516	7.3	49	608	20%	1894
PDP-11/24	2786	9.7	43	567	13%	1763
PDP-11/44	2786	9.7	43	733	15%	1764

A comparison of the initial and current versions of R1.

Figure 2.

individuals would collect the needed knowledge, add it to the system, and depending on the press of other problems, do more or less testing to determine that the overall capability of the system had not worsened. As time passed, the developmental process acquired substantially more structure. Planned release dates are now preceded by extensive testing of the system.

The article describing the initial version of R1 (McDermott, 1982) provides some insight into the nature of R1's knowledge by presenting a variety of measurements. Figure 2 compares the measurements from the initial version of R1 with corresponding measurements from the current version. Since a significant amount of the knowledge in the current version is specific to just a subset of the system types it can configure, Figure 2 provides the measurements for system-specific configurers as well as for the union of those configurers. Until recently, instead of a single version of R1 that could configure all system types, there was a version of R1 for each system type. Each of these versions consisted of a set of from 50 to 100 rules specific to a system type and two much larger sets of rules; it shared one of these rule sets with all of the other system types and the other with the system types having the same primary bus. About 300 of the shared rules were themselves specific to just one of the system types; each of these rules was included with the shared rules because it was relevant to a shared subtask.

R1's rules are grouped together on the basis of the subtask to which they are relevant; the "number of rules" column displays the total number of rules available to R1 in performing the configuration task, and the "average number of rules per subtask" column displays the mean number of rules in a group. The 3303 rules the current R1 has is the union of the rules of each system-specific configurer; the 10.3 rules per subtask is the union of the groups of rules the system-specific configurers bring to bear on a particular task. The "average number of parts ordered" column displays the number of components R1 has to configure. This number is significantly larger than the number of components listed on a purchase order since those line items actually refer to bundles of configurable components.

The numbers in the "average rule firings" and "percent of knowledge frequently used" columns are based on small sets of runs. For the initial R1, the numbers came from running R1 on 20 typical orders. For the current R1, the numbers came from running each system-specific version of R1 on about 20 orders of comparable complexity. The "average rule firings" column shows that substantially more is done in configuring a VAX-11/780 order now than was done initially; almost twice as many rules are applied. Two factors contribute to this increase. The configuration task has been enlarged by definition (*i.e.* there is now more to do), and second, there has been an increase in the average number of components per order.[2]

The "percent of knowledge frequently used" column shows what percentage of the rules are used at least once in at least one of the sample runs. Thus for the initial R1, 44% of the 777 rules were applied at least once over the 20 sample runs, and for the current R1, 47% of the 3303 rules were applied at least once over the approximately 200 sample runs. The fact that a substantial fraction of R1's knowledge is used only rarely is, of course, just what we would expect of a knowledge-based system. But the percentages for the system-specific versions are somewhat misleading. We would expect the percentage for each version to be lower than the overall percentage because each was run on only about 20 orders. However, because each version has knowledge that is not relevant to its tasks, the percentages for the versions are lower than they otherwise would be. The percentages for the VAX-11/780, the VAX-11/750, and the VAX-11/730 are the most accurate, but even they are too low by several percentage points. Since the nature of the knowledge used by

[2]On the average, 1.67 VAX-11/780 cpu minutes are required to configure an order.

each version is quite similar, it is likely that the percentage of the knowledge frequently used by each is pretty much the same—somewhere between 35% and 40% .

About 65% of the 2526 rules added to R1 since 1980 extend R1's general configuration capabilities; only about 35% of the rules are specific to a single system type. Of the 65% at least 15% were added to correct or refine knowledge of how to perform some subtask. This lower bound is suggested by the fact that the "average number of rules per subtask" increased by 30% during the past four years (*i.e.*, about 230 rules were added to the groups of rules applicable to the subtasks the initial R1 knew how to perform); adding a rule to the group applicable to some subtask is almost invariably done to correct or refine the knowledge of how to perform that subtask. The 15% is a lower bound because as the knowledge required to perform some subtask grows, it may become evident that what was viewed as a single subtask can be viewed as two or more simpler subtasks; what we do not know is how much the average number of rules per subtask would have grown if this subtask splitting had never occurred.

The Kinds of Changes R1 Has Undergone

As it turned out, the task of developing R1 had just begun when it was first put into use. In this section, we attempt to give a flavor of the kinds of changes that have been made to R1 over the past four years by examining a few examples in some detail. Our primary purpose in examining the growth of R1's knowledge is to better understand what is involved in adding knowledge to such a system. We can identify four reasons why knowledge was added to R1:

- To make minor refinements (adding knowledge to improve R1's performance on an existing subtask);
- To make major refinements (adding the knowledge required for R1 to perform a new subtask);
- To extend the definition of the configuration task in significant ways.

Ordinarily when people talk about why knowledge is added to an expert system, they seem to have the first reason in mind. As we have seen, of the more than 2500 rules added to R1 during the past four years, the data in Figure 2 suggest that more than 10% have been added to make minor refinements, fewer than 40% have been added to make major refinements, at least 35% have been added to provide functionality needed to deal with new system types, and perhaps as many as 15% have been added to extend the definition of the task in significant ways.

Minor Refinements. A knowledge addition of the first type is required when R1 cannot perform some subtask that it was thought to be able to perform. For example, over the years R1 has made several errors involving the placement of backplanes in boxes. One instance of such an error has to do with a backplane's location. In one variety of a 24 slot box, because of power considerations, a backplane is not permitted to cover slot 10. R1 knew that if it covered slot 10 when placing a backplane, it needed to move that backplane toward the back of the box so the backplane's front edge would be in slot 11. R1's knowledge was incomplete because it did not know it had to move any previously placed backplane from the front of the box toward the middle so that its back edge would be in slot 9. This backplane has to be moved toward the middle because leaving a larger space between the two backplanes would mean the standard cable used to connect backplanes could not be used (since it is not long enough). Fixing R1 was a straightforward task, but it required a certain amount of creativity (*i.e.*, it was not just a matter of "adding some more domain knowledge.") What R1 lacked was any notion of "deliberately vacant space." In order to provide rules that could recognize situations in which blank space was inappropriately positioned, R1 had to have the concept of blank space and an understanding of how to make a note that a particular space had been left blank on purpose. Given this, it was straightforward to add a few rules that recognized when some piece of blank space was inappropriately located and swap it with a backplane.

Major Refinements. A knowledge addition that results in a major refinement to R1 can be made in two kinds of situations: when R1 does not have any knowledge about how to perform some subtask, and when its knowledge of how to perform some subtask becomes so tangled that ways need to be found of representing the knowledge more generally. Brief examples of both situations are presented below; in the following section we provide a more lengthy analysis of one attempt to rewrite a set of rules, initiated almost purely to increase generality and understandability.

Most of the modules R1 configures on a UNIBUS consist of one or more boards that plug into backplanes which go in boxes. If multiple boards are required, they are usually placed next to each other in the same backplane. A situation unfamiliar to R1 arose when a module was designed with boards on two buses. Its first board was to be configured in an SPC backplane while the three remaining boards were to be configured in a special backplane that had to be located in the same box as the first board, but not in the same backplane. One way of extending R1 to handle this new component would have been to use a look-ahead strategy; R1 would have checked for space, power, and cabling constraints on the special backplane before configuring the first board. An alternative would have been a simple backtracking strategy. The approach R1 actually took involved a combination of both look-ahead and backtracking. R1 applies the same rules it uses for other modules to configure the first board; a few special rules then try to foresee abstract constraint violations involving the rest of the boards. If a problem is found, the first board is unconfigured. If no constraints are violated, power and space are reserved for the remaining boards.

Early in R1's history, only two types of panels needed to be considered. A few rules were sufficient to guard against the possibility of trying to configure two panels in the same space in a cabinet. Templates were used to describe panel placement possibilities; the rules recognized when some particular space was already occupied and avoided that space. As Digital introduced new products the situation became increasingly complicated until five different types of panels as well as disk drives and boxes could potentially occupy the same space with differing degrees of overlap. Because the original approach required all possible conflicts to be enumerated, it became increasingly unwieldy as the problem grew in complexity. The new solution involved redesigning the templates so the information they contained could be manipulated by a small number of more general rules and by making minor changes to the action parts of about 60 already existing rules that dealt with cabinet space decisions. This strategy worked well for about a year until Digital redesigned its cabinets to comply with new FCC regulations. At that point, the templates became too unwieldy because of the sheer number of possible individual locations; since the redesign also eliminated most of the irregularities of the previous problem, it became possible to simplify the templates and keep track of potential conflicts with a few very general rules.

New System Types. Providing R1 with the functionality it needed to deal with new system types has constituted a significant portion of the development effort. Since major configuration differences exist among the various buses supported by different CPU types, it was not clear initially how much configuration knowledge is common across system types. When a VAX-11/750 configurer was developed, the VAX-11/780 configurer was used as a model, but the knowledge bases were initially completely separate. Once the VAX-11/750 configurer had enough knowledge to be useful, it was merged with the VAX-11/780 configurer. On the other hand, the VAX-11/730 configurer was integrated, from the beginning of its development, with the older R1; the new version was developed by creating a small knowledge base (consisting of about 100 rules) specific to the VAX-11/730, adding some rules specific to the VAX-11/730 to the common knowledge base, and generalizing several of the rules in that common knowledge base. This approach worked well for the VAX-11/730, but when we turned our attention to the PDP-11/23+, we reverted to the approach we had used for the VAX-11/750. Several factors were involved in this decision. R1, up to this point, knew only of VAX-11 systems, which are UNIBUS and MASSBUS based, while the PDP-11/23+ is based on the LSI22 bus. The rules for configuring these buses have little in common. Moreover, the PDP-11/23+ supports a variety of operating systems, requires a completely different paneling structure, and assumes different power and capacity characteristics for its boxes and backplanes. Since the PDP-11/23+ is quite dissimilar to the VAX-11 systems, a separate version of R1 was developed for this task. Each of the subsequent system configurers was integrated with either the VAX-11 or the PDP-11/23+ system (depending on whether it had a UNIBUS or a LSI22 bus). Recently, it was decided that in a production environment, it would be advantageous to have one single system; by April 1984, all of the system configurers had been merged. Future system type additions will be part of this single version of R1 from the beginning of their development.

Adding the knowledge required to deal with new system types is non-trivial even when the new type is quite similar to types R1 already knows how to configure. Part of the effort of extending R1's configuration capabilities to cover a new type is due simply to the added amount of knowledge. For each of the types, we have had to add a great deal of data to the data base as well as make extensive rule changes and additions. Many of the decisions involve how to represent the new knowledge in the rules, but new data base representations are sometimes also required. The full extent of the effort varies, depending on the degree of similarity between the added system type and the types R1 can already configure. When there is a high degree of similarity, the form in which the existing knowledge is represented provides substantial guidance for how to represent the new knowledge. When the new system is quite dissimilar, substantial amounts of design are required.

Extending the Task Definition. As R1's role in Digital's manufacturing process has evolved, knowledge has been added to R1 that extends the definition of its task. For example, R1 was extended in January of 1983 to handle "multiple-CPU" orders. R1 was originally designed to deal with orders containing a single CPU. But multiple-CPU orders have become increasingly common, especially with the advent of smaller system types where multiple identical systems and/or several different systems on the same order are the norm. Part of the challenge of extending the definition of R1's task involves finding a way to realize some new capability that does not require extensive modifications to R1. In this case, we avoided the temptation of trying to modify R1 to configure multiple, loosely coupled systems simultaneously. Instead, a few new rules (originally about 10) were written to group the components into individual systems; each system was then configured in turn. Changes had to be made to 5 existing rules that determine what to configure and what order information to save; a few external initialization and output routines also had to be modified. The hard part was determining how R1's task definition could be extended most simply.

A substantial change to R1 in July of 1982 modified it to deal with a different categorization scheme for components. The component descriptions had been developed exclusively for R1 and were tailored to the configuration task. As Digital developed other knowledge-based systems for other purposes, it became desirable to have a common data base, where the components were categorized in a less ad hoc fashion. Before R1 could use the new descriptions, nearly all of its rules (about 2000 at the time) had to be changed, and for several hundred of these rules, the task of reformulation took considerable thought.

While the difficulty of making changes of any of the four types we have just described is highly dependent on the nature and scope of the knowledge that needs to be added, it also appears to be dependent on the amount of knowledge the system already has. In the early days, when R1 was small, people who joined the project were able, reasonably quickly, to acquire enough of an understanding of the configuration task and of R1's approach to it to become competent developers. But now that R1 has grown substantially, its sheer magnitude seems to serve as a barrier to the would-be developer. It takes much longer now for someone who joins the group to gain an adequate understanding of how R1 does configuration.

A Change over the Years

To provide another view of R1's development, we have analyzed the changes in R1's knowledge for two closely related tasks. One of the tasks involves deciding what backplane should hold the next set of modules. The other is a subtask that may or may not be performed depending on what the backplane selection possibilities are. The decision of what backplane to configure next is constrained by the pinning type of the modules, the space and power available for them, the current length of the bus and its loading, and the number and mix of backplanes that have been ordered. A good backplane choice is one that minimizes the number of additional components that have to be added, while satisfying all the constraints. The subtask is performed if the pinning type of the next module to be configured is SPC. In this case, two different sized backplanes could be used, so R1 must do some analysis of the implications of selecting each. Figure 3 shows how R1's knowledge of these tasks has developed; the development can be viewed as a series of minor refinements, followed by a major refinement.

In December 1980, R1's knowledge of how to perform the two tasks consisted of 36 rules, 23 rules for the selection task and 13 rules for the subtask. In October 1983, R1's knowledge consisted of 73 rules, 54 for the selection task and 19 for the subtask. During the intervening three years, 40 rules were added, 3 rules were eliminated, and 11 rules were changed. This alteration is consistent with the knowledge-based approach, where the initial instinct is to solve a problem by adding more knowledge. It suggests that the rules eventually formulated are for the most part adequate, but that it takes a long time to collect the relevant knowledge. The fact that only 11 rules were changed may be a little misleading since 27 of the added rules were special cases of existing rules, implying that the conditional part of many of the unchanged rules were inadequately discriminating. Of the rules that were changed, the changes were almost all in the conditional part and were in the direction of making the rules increasingly discriminating.

In October 1983, one of the people working on R1 observed that if R1 were given more knowledge of how to assess the likely implications of various decisions, it would need to backtrack even less often. In the course of reworking this capability, the number of rules remained constant, but the

	12/15/80		10/15/80		12/15/80
TOTAL RULES	36		73		73
Rules Added		40		31	
Rules Deleted		3		31	
Rules Changed		11		32	
Condition Elements Added		11		8	
Condition Elements Deleted		1		72	
Condition Elements Changed		15		40	
Action Elements Added		1		5	
Action Elements Deleted		0		8	
Action Elements Changed		1		13	

Two sample subtasks

Figure 3.

level of expertise improved dramatically. Of the rules in the October version, 31 were eliminated and, coincidentally, 31 were added; of the remaining 42 rules, 32 were changed. Again, this alteration is what we might expect of a situation in which a capability is being substantially extended. When the knowledge is all laid out and it is clearer what other pieces of knowledge are relevant to the task, it becomes more obvious how to represent the knowledge cleanly. In this case, the biggest change was the elimination of condition elements. This happened because it became clear that the rules were too constraining; that is, the rules had typically been added to deal with a particular error, and so the October version had a small set of overly general rules (from the initial version) and several more overly specific rules. Seeing all the knowledge laid out made it possible to hit the right level of specificity.

Conclusions about Growth

The following conclusions purport to provide guidance to the developers of any knowledge-based application system. We are of course not at all sure what aspects, if any, of the experience with R1 at Digital will turn out to be typical. It seems reasonable to believe, however, since R1's task is knowledge-intensive, that the experience with R1 relating to the rate at which it has acquired knowledge and the difficulty of adding that knowledge will at least have relevance to other attempts to put knowledge-based systems to work on real tasks.

Even though the experts claimed in 1979 that R1 had most of the knowledge it needed, a great deal of knowledge has been added to R1 over the past four years. There is no more reason to believe now than there was then that R1 has all of the knowledge relevant to its configuration task. This, coupled with that fact that R1 deals with an ever-changing domain, implies its development will never be finished. Thus users of systems like R1 will have to be emotionally prepared to interact with a less than perfect program. They will have to be as forgiving of ignorance in these expert systems as they are of ignorance in humans who are ever becoming more expert.

Though much of R1's knowledge was added to correct or complement existing knowledge, a significant part of the additions came as a result of R1 having to have the knowledge to perform new tasks. Some of these were the result of Digital introducing new computer system types and the rest resulted from the users' observations that things would be better if R1 could do one more thing. We believe all expert systems will be hounded to continue to grow for both of these reasons. Tasks that expert systems are good for are just those whose objects change significantly over time. Moreover, in such tasks no clear boundaries delimit what should and should not be within the province of the expert. Thus, whenever an expert system finds itself on a boundry, its public encourages it to extend the boundary.

Situations arise in which the task of adding a piece of knowledge is *extremely* straightforward because the new knowledge needs to be represented and used in virtually the same way as the system's existing knowledge. But, for the most part, adding a piece of knowledge involves some amount of creativity. In domains other than configuration (or at least in diagnostic as opposed to constructive tasks) domain knowledge appears to be substantially more regular and can be added routinely. Significant, but as yet undiscovered, regularities in configuration knowledge may exist that will someday allow it to be added more easily. But for now, it is important to at least be open to the possibility that a knowledge-based system will forever have to be surrounded by people who know how to do development. They will be called upon to be innovative and adaptable. Although it may be the case that adding knowledge incrementally is easier than rewriting or modifying a traditional program, by no means can this task be done without substantial amounts of problem solving.

It was clear before R1 was a year old that the incremental addition of knowledge resulted in a system with a significant amount of redundancy and a penchant for *ad hoc*ery. To the extent that adding knowledge to the system involves human intervention, this general lack of cleanliness and conciseness provides an obstacle to the system's further development. Few expert systems are likely to be redeveloped (as R1 was in 1980, but not since). However, we suspect that from time to time, some part of every expert system's knowledge will become so convoluted that its developers will take the time to re-represent that knowledge.

R1's Performance

Before R1 began to be used, each system Digital received an order for was configured by a technical editor, typically on the day before the system was to be assembled and tested. The technical editor examined each order to determine whether configuration constraints required additional or different components and then specified some of the relationships among the components on the order. Though the task was performed at a fairly high level of abstraction, it seldom took fewer than 5 or 10 minutes to configure an order, and complex orders took substantially more time. When R1 began to be used, it essentially became a technical editor. But since it was not clear initially how well R1 was going to do as a technical editor, some of the people who had been technical editors stayed to watch over R1. In effect, they became R1's mentors. Every order configured by R1 has been examined, more or less closely, by a mentor and if the mentor believed the configuration was lacking in any respect, he or she reported the problem to the R1 development group.

Although R1 is an expert system in the sense that the body of knowledge it uses to perform the configuration task is acquired by human experts over a period of years, its task is different from the task that used to be performed by the technical editors because R1 configures systems at a significantly greater level of detail than they did. Because its task is more extensive, it is hard to answer the question: Does R1 do as well at the technical editing task as human experts do? The task R1 actually performs is the old technical editing task plus part of the task performed by the technician who physically assembles the system (since the technician has to descend to R1's level of detail to do his job). But the technician's situation is different from R1's in that the technician has the physical components that need to be assembled and tested in front of him and can discover when components are missing or misconfigured in more direct ways than are available to R1. Thus we have not tried, in this article, to compare R1's performance with that of the human experts. The closest we come to examining that relationship is with the bogus problems category. A bogus problem is one that a human expert reports as an R1 error, but that on further examination turns out to have been a failure on the part of the expert to appreciate correctness.

The data presented in this part of the article leave something to be desired; part of the problem is that it was not clear, at any point during the past four years, how frequently R1's performance needed to be sampled. Since knowledge-based systems continue to be developed incrementally as they are used, it was obvious that collecting performance data would be an integral part of using the system. It was also clear that the more data that were collected, the better we would understand the extent to which R1's knowledge was incomplete. But all that is really required to drive the developmental process is enough data to give the people collecting and encoding R1's knowledge plenty to do. Since finding inadequacies in R1's knowledge has never been very hard, more attention was given to the task of extending and refining R1 than to the data collection task. As a result, there are a few periods, in two cases extending for months, in which the data we have are incomplete. For the most part, however, we have some information about how well R1 performed on each order it configured.

Even if we had information about each order R1 configured, our data would still be unsatisfactory because our understanding of how to collect the relevant data has grown slowly. Since people who have the responsibility of reviewing each of R1's configurations have little understanding of how R1 does what it does and where and how it can err, they can only report error manifestations. Devising a process that makes it fairly straightforward to link manifestations to causes (so, for example, the number of instances of each error type can be determined) took some time. Initially the process used paper and pencil. A second issue, then, was how to design a program that could assist with the data collection task. Because it took time to devise such a program (a lot of time since it had low priority), a significant part of our task has been to reconstruct, from incomplete descriptions of error manifestations, what the actual errors were. We feel relatively confident in the overall results, but are sure a number of minor inaccuracies exist.

Before presenting the performance data, we need to discuss briefly how "percentage of totally correct orders" came to be accepted early as the metric for measuring R1's performance. The problem with this metric, of course, is that it does not discriminate between terrible performance (gargantuan errors) and near perfect performance (tiny, almost insignificant errors). In retrospect, it is clear that having some idea of the seriousness of each error would be helpful in evaluating R1. But when R1 first started to be used, it was with the expectation that there were only a few things it did not yet know, and the only question in people's minds was how many weeks it was going to take before R1 knew everything. Within that context, it is not at all surprising that the all or nothing metric was selected; anything else would have seemed too fine-grained.

Some Performance Data

Figure 4 provides a detailed account of R1's performance over the past four years. The information is presented by quarter, beginning in January 1980 and ending in December 1983. Three major problem categories exist: rule problems, data base problems, and other problems. For rule and for data base problems, as well as for total problems, the percentage of orders containing that type of error is given. Within each category, information is provided about one or more subcategories. For each subcategory, the number of problem instances as well as the number of distinct problems is reported. The total problem instances percent gives a sense of R1's usefulness. However, since most errors R1 now makes are minor, its output, even if there are problems, can usually be used, though sometimes only after a bit of editing. The distinct problems percent in the parts and rules subtotal gives a sense of R1's competence; this measure shows the number of distinct errors R1 has made due either to missing or incorrect configuration knowledge or to missing or incorrect component descriptions. Few, we think, would want to claim that R1 was a competent configurer during its first year of use; but for the past two years, its lack of knowledge has been well within the bounds of respectability. The number of problem instances divided by number of distinct problems gives an indication of how many times a problem occurs before it is fixed.

The most significant improvement in R1 has come in the percent of problems attributable to missing or incorrect rules. While missing or incorrect domain knowledge has never been the most significant source of problems, it is now the case that fewer than one in a thousand orders is misconfigured because of rule problems. One might ask (though we hope only in jest) how after four years R1 can have any missing or incorrect domain knowledge. There are at least two answers. First, even though R1 has configured more than 80,000 orders, it has seen only a small fraction of the situations it could possibly encounter. Second, new products are sometimes announced before R1 acquires all the knowledge it needs to be able to configure those new products correctly.

Problems with parts have been much more troublesome. Incorrect part descriptions have never been much of a problem, but missing part descriptions have been a significant problem during all four years. During the first two years R1 was used, the reason it was sometimes given systems to configure containing components not described in its data base had mostly to do with the fact that the people responsible for adding part descriptions to the data base were not the right people. It was assumed initially that the component descriptions could be created by people who knew a lot about the components, but knew little about how R1 would use the descriptions. As it turned out, creating useful descriptions is not all that straightforward. It often is not clear what "configuration level" means, not clear what attributes are required, and not clear what knowledge to put in the rules and what in the data base. In order to know what information a description should contain, it is necessary to know how the information is going to be used. In order to know how the information is going to be used, it is necessary to know something about the component. After trying various strategies for making the middle-men more productive, the responsibility for creating descriptions was taken over by the people who encode the configuration knowledge in rules.

This change would have solved the missing part descriptions problem were it not that at about that time, the number of orders R1 was configuring per quarter began to increase substantially. As a consequence, the number of different parts ordered grew significantly. Since R1 has descriptions of only 5,500 of the more than 100,000 parts that could appear on an order, and since the rate at which the as yet undescribed parts appear on orders is very low, the development group adopted the strategy, for low volume parts, of waiting until the part shows up on an order before adding its description to the data base. This policy is less cavalier than it may seem since when one of these low volume parts does show up on an order, it usually turns out to be a part that is not itself configured (*e.g.*, software or an accessory). Thus although any configuration mentioning a part R1 does not know about is counted as a problem, most of the time those configurations can be used without modification.

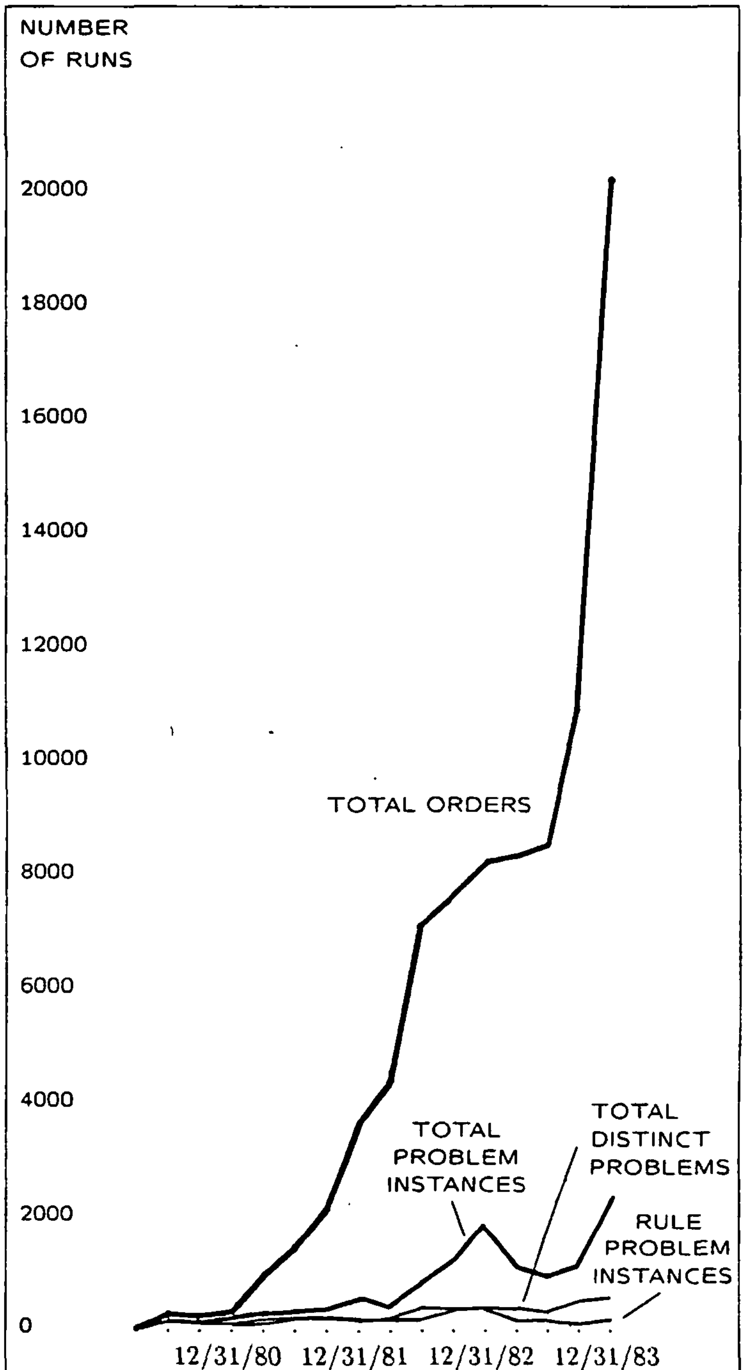
	1st Qtr 1980	2nd Qtr 1980	3rd Qtr 1980	4th Qtr 1980	1st Qtr 1981	2nd Qtr 1981	3rd Qtr 1981	4th Qtr 1981	1st Qtr 1982	2nd Qtr 1982	3rd Qtr 1982	4th Qtr 1982	1st Qtr 1983	2nd Qtr 1983	3rd Qtr 1983	4th Qtr 1983
Incorrect Rules																
Problem Instances	6	21	12	26	55	-	-	171	123	313	243	342	100	58	13	80
Distinct Problems	6	7	8	13	18	-	–	30	25	41	38	37	16	14	6	19
Total Orders	54	194	133	210	824		-	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem Instances Percent	11.1%	10.8%	9.0%	12.4%	6.7%	–		4.7%	2.9%	4.4%	3.2%	4.2%	1.2%	0.7%	0.1%	0.4%
Distinct Problems Percent	11.1%	3.6%	6.0%	6.2%	2.2%	–	–	0.8%	0.6%	0.6%	0.5%	0.5%	0.2%	0.2%	0.1%	0.1%
Missing Part Descriptions																
Problem instances	6	51	16	22	33	–	–	136	78	214	629	997	697	535	906	1962
Distinct problems	6	44	16	19	31	–	–	43	42	40	105	157	162	141	368	472
Orders with errors	4	21	7	12	19	-	-	104	65	166	439	601	535	464	920	1275
Incorrect Part Descriptions																
Problem instances	1	9	17	15	21	–	–	74	31	86	94	86	23	19	10	28
Distinct problems	1	3	3	6	12	–	-	12	11	16	9	19	8	6	5	9
Parts Subtotal																
Problem instances	7	60	33	37	54	-	–	210	109	300	723	1083	720	554	916	1990
Distinct problems	7	47	19	25	43	-	–	55	53	56	114	176	170	147	373	481
Total orders	54	194	133	210	824	–	–	3605	4283	7100	7503	8110	8192	8247	10775	20241
Problem instances percent	13.0%	30.9%	24.8%	17.6%	6.6%	–	-	5.8%	2.5%	4.2%	9.6%	13.4%	8.8%	6.6%	8.5%	9.8%
Distinct problems percent	13.0%	24.2%	14.3%	11.9%	5.2%	–	-	1.5%	1.2%	0.8%	1.5%	2.2%	2.1%	1.7%	3.5%	2.4%
Parts and rules Subtotal																
Problem instances	13	81	45	63	109	116	190	381	232	613	966	1425	820	612	939	2098
Distinct problems	13	54	27	38	61	–	–	85	78	97	152	213	186	161	384	509
Total orders	54	194	133	210	824	1304	2040	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem instances percent	24.1%	41.7%	33.8%	30.0%	13.3%	8.9%	9.3%	10.5%	5.4%	8.6%	12.8%	17.6%	10.0%	7.3%	8.7%	10.4%
Distinct problems percent	24.1%	27.8%	20.3%	18.1%	7.4%	–	–	2.3%	1.8%	1.4%	2.0%	2.7%	2.3%	1.9%	3.6	2.5%
Operational Problems																
Problem instances	2	45	16	56	77	115	67	12	7	2	41	0	11	1	0	0
Controversial Issues																
Problem instances	0	1	3	3	9	11	47	80	21	33	19	26	50	46	19	25
Distinct problems	0	1	2	1	3	–	–	4	3	8	9	5	8	7	4	4
Desired Enhancements																
Problem instances	–	–	–	–	–	–	–	38	6	19	11	57	86	155	25	27
Distinct problems	–	–	–	–	–	–	–	3	2	6	1	2	5	7	3	4
Bogus Problems																
Problem instances	0	3	1	7	27	16	31	15	15	36	76	62	44	23	33	43
Total Problem Reports																
Problem instances	15	130	65	129	222	258	335	526	281	703	1113	1570	1011	837	1016	2193
Distinct problems	15	103	46	102	168			119	90	149	279	282	254	199	424	560
Total orders	54	194	133	210	824	1304	2040	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem instances percent	27.8%	67.0%	48.9%	61.4%	26.9%	19.0%	16.4%	14.6%	6.6%	9.9%	14.8%	19.3%	12.3%	9.9%	9.4%	10.8%
Distinct problems percent	27.8%	53.1%	34.6%	48.6%	20.4%	–	-	3.3%	2.1%	2.1%	3.7%	3.5%	3.1%	2.4%	3.9%	2.8%

R1's Performance by Quarter

Figure 4.

The problems not really under the control of R1's developers—operational problems, controversial issues, desired enhancements, and bogus problems—have always been a significant part of the problems reported. During the first year and a half, a very large fraction of the problems were operational; a number of factors, each by itself not very significant, conspired to separate R1 from its user community. During 1983, the number of bogus problem reports grew to become a highly encouraging (from R1's point of view) fraction of the total problem reports; from July through September, the number of bogus problem reports was actually double the number of rule problem instances, and during the other three quarters the number of bogus problem reports was about half the number of rule problem instances.

Figure 5 presents some of the information from Figure 4 in graphical form. The relationships among "total orders", "total problem instances", "total distinct problems" and "rule problem instances" are depicted. The "total orders" measure provides a context within which the error measures can be understood. The "total problem instances" measure provides a lower bound on R1's usefulness. The area under that curve indicates the number of orders for which R1's output was possibly not useful; however, as we have seen, in most cases the output could be used, though sometimes only after being modified. The "total distinct problems" measure provides a lower bound on R1's competence. The area under that curve indicates the number of different kinds of situations R1 did not deal effectively with. The "rule problem instances" measure indicates the extent to which R1's failures were due to its ignorance of the domain.



R1's Performance by Quarter (in Graphical Form)

Figure 5.

Conclusions about Performance

As in the previous section where some conclusions about growth were presented, the following conclusions purport to provide guidance to the developers of any knowledge-based application system. Since the conclusions we offer here are not very startling, it is quite likely that they have some general validity. All they really contain is the notion that when AI tools confront real tasks, the world is not going to obediently conform to all of the hopes of the tool maker. The real world treats AI tools with the same disrespect with which it treats all other tools and thus a great deal of the effort of bringing AI systems into regular use on real tasks involves doing things that do not have any special relationship to AI. What undoubtedly makes matters worse for AI tools is that the problems they are used to solve are ordinarily more open than the problems traditional software tools typically address.

In the previous section we argued that an expert system will never have all the knowledge it needs. Thus it will always make mistakes, and it is important for both the developers and the users to expect them. R1's performance data suggest something even stronger: To expect anything close to perfection during the first few years a system is being used (especially if the task is significantly more than toy) is probably a very serious mistake. We believe the data also suggest that to keep an expert system from regular use until its knowledge is complete would be a poor idea. It has taken 80,000 orders to uncover some of the inadequacies in R1's configuration knowledge, and the configuration task is continually redefined as new products are introduced. These facts suggest that even if someone had the time and energy to try to create a near perfect system before introducing it into production, many inadequacies would become evident with regular use.

It would be a mistake to believe the major or even a primary source of error in the performance of an expert system will be due to incorrect or missing domain knowledge. Depending on the number and type of objects the system is intended to deal with, large amounts of effort may be needed to collect and maintain the information about these objects. But even if the nature of the task makes data collection and maintenance relatively unproblematic, a variety of other sources of error may spring up as the system begins to be used. As we just mentioned, there is nothing magic about knowledge-based systems that allows them to avoid the problems other software systems have to face. Indeed, the fact that they continue to be developed while they are being used undoubtedly intensifies those problems. The relative seriousness of the various problems that confronted R1 would surely have been better appreciated if R1 had had a sophisticated problem reporting mechanism from the beginning.

If one looks at R1's performance over the first two years of its use and tries to imagine R1 being used in a situation where it was being asked to configure thousands of orders a month, it seems clear that its use would have been discontinued. This judgment is perhaps overly harsh since, as mentioned above, a significant portion of the configurations with errors could be used with only minor modifications. In any event, using R1 in a high volume environment would have made its initial nuturing substantially more difficult. R1 was used instead in an environment in which the initial demands on it were of the order of a few tens of orders per week for the first year. This small volume made it possible for people to jump in whenever R1 failed and to avoid depending too much on a system that at the time was far from being an expert in the domain.

Conclusion

One of our purposes in giving these glimpses of R1's developmental and performance histories is to provide some evidence for evaluating the claims that have been made about expert systems. Expert systems supposedly are easy to develop incrementally and, at some point, become as good as human experts. R1 lends some credence to both of these claims. While substantial effort has been required to develop R1, the approach taken has made it possible over a four year period to increase R1's knowledge substantially without starting over; this lends support to the first claim. The fact that human experts erroneously conclude that R1 has misconfigured systems about as frequently as R1 actually misconfigures systems lends some support to the second claim.

References

Forgy, C. L. (1979) OPS user's manual. Technical Report, Carnegie-Mellon University, Department of Computer Science.

Forgy, C. L. (1981) OPS5 user's manual. Technical Report, Carnegie-Mellon University, Department of Computer Science.

Gupta, A. and C. L. Forgy (1983) Measurements on Production Systems. Technical Report, Carnegie-Mellon University, Department of Computer Science.

McDermott, J. (1980) R1: An expert in the computer systems domain. *In Proceedings of AAAI-80, National Conference on Artificial Intelligence,* Stanford, California, 269-271.

McDermott, J. (1981) R1's formative years. *AI Magazine*, Vol. 2., No. 2, 21-29.

McDermott, J. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence* 19(1), 39-88. Also available as a CMU, CSD technical report.

Reading Guide:

You should have the following material:

"On the Development of Commercial Expert Systems"
(a paper discussing Dipmeter Advisor)

"RI Revisited: Four Years in the Trenches"

From Vol II of the Handbook of Artificial Intelligence:

Overview (pages 79-84)

TEIRESIAS example (pages 92-101)

PROSPECTOR (pages 155-162)

I suggest reading the first few pages of the overview, for historical background, followed by the Teiresias example, to give you an idea of what is possible as far as the human-computer interface.

Next, I think you should wade through the entire paper on Dipmeter Advisor. The first four pages will give you an idea of how a "current" geologic expert system works, while the rest of the paper will indicate what is involved in building a system. I'll admit the second half of this paper will contain alot of unfamiliar terminology and irrelevant discussion, but you should still be able to glean alot of useful information out of it.

The rest of the material will show the state-of-the-art as of a few years ago (the most recent general information in print) and some idea of system growth. The Rl paper, in particular, is only worth a light skimming.

The Handbook of Artificial Intelligence

Volume II

Edited by

Avron Barr

and

Edward A. Feigenbaum

Department of Computer Science
Stanford University

HEURISTECH PRESS
Stanford, California

WILLIAM KAUFMANN, INC.
Los Altos, California

Library of Congress Cataloging in Publication Data:

The handbook of artificial intelligence.

Bibliography: p. 381
Includes index.
1. Artificial intelligence. I. Barr, Avron, 1949–
II. Feigenbaum, Edward A.
Q335.H36 001.53'5 80-28621
ISBN 0-86576-004-7 (set)
ISBN 0-86576-006-3 (Vol. II)

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

A. OVERVIEW

OVER the past decade, many of the fundamental AI techniques described in the previous chapters on search, knowledge representation, and natural-language processing have been applied in the form of *expert systems*, that is, computer systems that can help solve complex, real-world problems in specific scientific, engineering, and medical specialties. These systems are most strongly characterized by their use of large bodies of *domain knowledge*—facts and procedures, gleaned from human experts, that have proved useful for solving typical problems in their domain. Expert-systems research promises to lead to AI applications of great economic and social impact. But far from being solely concerned with applying AI problem-solving techniques, the research described in this and the following two chapters has often addressed fundamental questions concerning the nature of knowledge, both in terms of formal representational systems and as an essentially social phenomenon—knowledge as something that must be shared and transferred among men and machines.

Evolution of Expert Systems

AI research in the 1960s identified and explored several general-purpose problem-solving techniques. This work introduced and refined the concept of heuristic search (see Chap. II, in Vol. I) as an important model of problem solving. Many of the AI systems developed during this period, like GPS, the Logic Theorist, REF-ARF, QA4, and PLANNER (all described elsewhere in the *Handbook*), dealt with problems in simple, constrained domains such as chess, textbook problems, robot planning, blocks-world manipulations, and puzzles like "Tower of Hanoi" and "Missionaries and Cannibals." But by the mid-1960s, some researchers in the DENDRAL project at Stanford and the MACSYMA project at M.I.T. had begun work on the first expert systems—organic chemical analysis in the case of DENDRAL and symbolic integration and formula simplification in MACSYMA.

These systems were designed to manipulate and explore symbolically expressed problems that were known to be difficult for human researchers to solve. The problems were characterized by the increasing number of solution possibilities that had to be examined as the problem specifications grew in complexity—the larger the size of the problem specification (e.g., the size of the molecule or the complexity of the expression to be integrated), the more difficult it was for human researchers to discover solutions or be confident that all valid solutions had been found. This *combinatorial explosion* in the solution search space often outstripped the abilities of human researchers. The capability of AI systems to deal with the larger solution spaces is important in that it extends the types of problems that can be solved with the same conceptual tools.

More recently, several other factors have motivated research on expert-systems development. Most notably, expert systems promise to be quite profitable because they can help solve hard problems that require the best (most expensive) human expertise. (See, e.g., Articles VII.C4 and VII.D3 on systems that may help design chemical-synthesis techniques and explore for mineral deposits.) In some domains, like medical diagnosis, the fact that the exhaustive nature of problem solving in expert systems ensures that remote possibilities are not overlooked is important. And often the very codification of expertise in suitable form for an expert system is an illuminating and valuable part of the expert-systems development. (This systematic reorganization of what is known can lead, e.g., to new insights into the structure of the domain or to new ideas about how to teach it.)

In a domain like medicine (and unlike symbolic integration) where the nature of the problem is not sufficiently understood to completely specify the search space, large amounts of domain-specific knowledge have to be represented and reasoned with. Thus, while heuristic-search management is still a major concern in the construction of any expert system, efficient implementation and automated maintenance of large knowledge bases must also be addressed. A particularly important design issue is devising effective means for acquiring such large amounts of knowledge from the human experts, who insist on "talking about" what they do rather than "dumping" what they know, as computers do.

The issue of acquiring knowledge from human experts is now seen as a part of the general problem of *transfer of expertise*. Since humans are both the *source* and the eventual *users* of expertise, current concerns in expert-systems design center on considerations of how humans talk about what they know. For an expert system to be truly useful, it should be able to learn what human experts know, so that it can perform as well as they do, understand the points of departure among the views of human experts who disagree, keep its knowledge up to date as human experts do (by reading, asking questions, and learning from experience), and present its reasoning to its human users in much the way that human experts would (justifying, clarifying, explaining, and even tutoring). These issues in the transfer of expertise can be seen as a microcosm of many of the central concerns of Artificial Intelligence.

Representing Expertise

Specialists are distinguished from laymen and general practitioners in a technical domain by their vast task-specific knowledge, acquired from their training, their subsequent readings, and especially their experience of many

A

hundreds of cases in the course of their practice. Whether car mechanics or neurosurgeons, experts can solve problems that others cannot, because they know things that nonexperts do not. Sometimes this knowledge is in the form of specific facts about the domain that have, over the years, been committed to memory, and sometimes the expertise appears as hunches, "educated guesses" about the way to proceed in problem solving.

Representing and using the various types of knowledge that characterize expertise constitute one principal focus of expert-systems research. Among the things that might be useful for an expert system to know about are:

1. Facts about the domain: "The shin bone is connected to the ankle bone" or, more typical of human experts, "The automatic choke on '77 Chevys often gets stuck on cold mornings";
2. Hard-and-fast rules or procedures: "Always unplug the set before you stick a screwdriver into the back";
3. Problem situations and what might be good things to try to do when you are in them (heuristics): "If it won't start but you are getting a spark, check the fuel line";
4. Global strategies: differential diagnosis;
5. A "theory" of the domain: a causal explanation of how an internal-combustion engine works.

All of the knowledge-representation schemes described in Chapter III (in Vol. I) have been used in expert systems; in fact, much original work on knowledge representation has been done in the context of expert-systems design.

Note that much of the knowledge that characterizes human expertise is hunchlike, in the sense that it does not constitute definite consequences of actions or certainty of conclusions. Reasoning with such knowledge has been the key idea that made expert systems possible and constitutes the main problem in developing their power further. In particular, *inexact reasoning*, using hunches or *heuristics* to guide and focus what would otherwise be a search of an impossibly large space (see Articles II.C3 and II.C4, in Vol. I), has resulted in systems with human-level problem-solving abilities. Indeed, these systems have at times proved superior to the human experts, primarily because they consider a much larger set of possible solutions (as much as several orders of magnitude larger) and do not miss unlikely or unexpected possibilities, once these have been noted as worthy of consideration by the expert who built the knowledge base.

Transfer of Expertise

Solving real-world problems at human-expert levels of performance is only the beginning of expert-systems design. Most of the applications systems described in this chapter can be viewed as *consultants* that formulate opinions and give advice to their users. The tasks these consultants are designed to perform require the application of facts and relationships known only by specialists. The current systems emphasize the cognitive abilities that support interaction with the user during problem solving, such as the ability to explain lines of reasoning or to acquire new domain knowledge interactively.

Typically, such a system will be considered "intelligent" if it meets the following criteria: (a) The system gives correct answers or useful advice, and (b) the concepts and reasoning processes it uses to solve the problem resemble those that the user might employ. This last concern has led to the design of systems that can explain their reasoning about a case, maintain a focused dialogue with a user when pursuing relevant facts and inferences about his (or her) case, and employ knowledge at the conceptual level of the user when solving and explaining both the problem and the system's solution. Successfully addressing these primarily *human-engineering* concerns has required many advances in AI. These abilities and developments are detailed for each system in the following articles (see especially Article VII.B).

Explanation and the opacity of knowledge. As mentioned previously, a major design issue for some of these systems, for the consultants in particular, is whether the system needs to explain its reasoning to a user. This capability is implemented primarily to convince users that the system's reasoning is appropriate and that its conclusions about a case are reasonable.

Sometimes the problem-solving expertise of the system is in a form that is not at all similar to the expertise that a human expert would apply to obtain the solution. For example, in the case of the DENDRAL programs, the generator of chemical-structure candidates employs a procedure for exhaustively producing possible structures based on various graph-theoretic notions that organic chemists who use the system are unlikely to know or care about. Thus, a major portion of the DENDRAL expertise resides in a procedure that is conceptually *opaque* to the typical user. The generator was developed because it was discovered that the method used by chemists to find solutions for these problems is, in fact, incomplete, while the method used by the DENDRAL program has been mathematically proved to be complete. A similar situation exists in the MACSYMA system, which uses the Risch algorithm for evaluating various types of integrals. While mathematically correct, the algorithm is rarely employed by human mathematicians because of its complexity. The correctness and continuing success of these programs serve as their primary form of explanation: The user community is thereby convinced that the performing system is both acceptable and usable.

In contrast, *consultation systems* like MYCIN and PROSPECTOR have been designed to represent and explain the reasoning process of the system in a manner that is understandable to the knowledgeable user. These systems require a representational formalism capable of supporting the reasoning and explanation abilities that would closely approximate the conceptual framework of the expert and the user. Since most of these scientific and technical domains have a well-defined set of concepts that their practitioners use consistently, the systems' designers have capitalized on this consistency and have designed the programs to accept and reason with knowledge using these concepts.

Assuming that a system has an explanation facility, the system designer faces another issue: Should the system reason and apply the expertise in a manner that resembles the methods of human experts? In MYCIN, for example, no claim is made by the designers that the simple backward-chaining reasoning methodology has any resemblance to the methods actually employed by human physicians in diagnosing infectious diseases. Although the medical concepts employed by the system are familiar to most physicians, the method of inferring the infections and causal organisms, while understandable by physicians, bears little resemblance to their normal diagnostic reasoning. By contrast, the PIP and INTERNIST systems emphasize the similarities of their diagnostic procedures to those of physicians.

Knowledge acquisition. During the development of the knowledge base, experts are unlikely to present all of the relevant facts and relationships for expert performance in the domain. Being human, experts tend to forget or to simplify details about their knowledge, requiring the system to augment its knowledge at a later time. Since the knowledge imparted to the system is largely empirical and the domains are themselves developing rapidly, it is necessary for the system to make these changes easily and in an *incremental* or modular fashion. Thus, most of the recent applications systems have emphasized representation schemes that allow for the incremental construction of the knowledge base.

Most researchers have approached incremental construction by means of *production-rule* knowledge representation. Each rule, and rule set, represents a "chunk" of domain expertise that is communicable to the user and that can be added to or deleted from the system's knowledge base with relatively constrained changes in the system's behavior (see Article III.C4 and the discussion of *modularity* in knowledge representation in Article III.A, in Vol. I). Thus, the system can be improved by modifying the knowledge base with new rule sets that deal with new subdomains. Furthermore, the production-rule formalism can directly accommodate the knowledge of the domain experts in the form that they most often communicate it—for example, "In this situation I suspect this problem and perform these tests."

The Status of Applications Research

The major domains of expertise that have been developed as applications systems include the diagnosis and treatment of various diseases (see Chap. VIII), the design of computer assistants for both the analytic and the synthetic aspects of organic chemistry (Sec. VII.C), interactive tutoring systems in education (Chap. IX), and assistants for performing advanced mathematics

(Article VII.D1). A number of other notable applications have been developed, including applications of AI to database information-retrieval problems (see Article VII.D4) and a geological prospecting assistant (Article VII.D3).

Among the rapidly growing host of applications-oriented systems are SACON, a system for advising structural engineers in the use of a large, finite-element analysis program for modeling various mechanical structures (Bennett et al., 1978); PUFF, a system for diagnosing a patient with pulmonary dysfunctions (Feigenbaum, 1977); and HEADMED, a system for diagnosing and treating psychiatric patients (Heiser, Brooks, and Ballard, 1978). More recent are McDermott's (1981) R1 expert on computer-system configurations and Stefik's (1980) work on an aid in designing experiments in molecular genetics (see also Article XV.D2, in Vol. III, on MOLGEN). Current research in this area includes extensions of the expert-system paradigm to computer-based assistants for computer-system failure diagnosis, aids for VLSI circuit design, more sophisticated database-query systems, and systems that can act as tutors in their areas of expertise (see Article IX.C6).

One important development in current research on expert systems is the emergence in recent years of "expert-systems-building" systems, which facilitate the construction of expert systems in any domain. For example, the EMYCIN system (van Melle, 1980) consists of the basic control structure of MYCIN, but with MYCIN's infectious-disease knowledge base removed. With another knowledge base substituted in the same production-rule format as MYCIN's, this "Empty MYCIN" system retains the capability of interacting with the user during a case, to explain its reasoning, and to answer questions about a case in the new domain. EMYCIN has been used successfully to develop the applications in the treatment of pulmonary dysfunction, in structural analysis, and in the psychiatric diagnosis mentioned earlier. Several other expert-systems-building systems are being developed, including IRIS (see Article VIII.B6), AGE (Nii and Aiello, 1979), OPS (Forgy and McDermott, 1977), and ROSIE (Fain et al., 1981; Hayes-Roth et al., 1981). Systems such as these, which attempt to facilitate the construction of expert systems, are an important area of current research.

Another primary research activity in the near future will be the development of better facilities for acquiring the domain concepts and the empirical knowledge that expert systems must have. Feigenbaum (1977) suggests that the painful process of *knowledge engineering*, which involves domain experts and computer scientists working together to design and construct the domain knowledge base, is the principal bottleneck in the development of expert systems. Efficient interfaces for acquiring this domain-specific knowledge, along the interactive transfer-of-expertise lines explored in TEIRESIAS (Article VII.B) or the automatic *theory-formation* methods used by the Meta-DENDRAL system (Article VII.C2c), need to be developed before significantly larger expert systems can be constructed.

missing from . . . . example of TEIRESIAS's use of rule models in its knowledge-acquisition dialogue is given in the sample protocol below.

Meta-rules and Performance Strategies

In performance programs with sufficiently small knowledge bases (like MYCIN's), exhaustive invocation of the relevant parts of the knowledge base during a consultation is still computationally feasible. However, with the inevitable construction of larger knowledge bases, exhaustive invocation will become unrealistic. In anticipation of this, *meta-rules* are implemented in TEIRESIAS as a means of encoding strategies that can direct the program's actions more selectively than exhaustive invocation can. The following meta-rule is from MYCIN's infectious-disease domain:

```
                    META-RULE 001
IF      (1) the infection is a pelvic-abscess, and
        (2) there are rules that mention in their
            premise Enterobacteriaceae, and
        (3) there are rules that mention in their
            premise gram positive rods,

THEN    There is suggestive evidence (.4) that the rules
        dealing with Enterobacteriaceae should be evoked
        before those dealing with gram positive rods.
```

This rule suggests that, since enterobacteria are commonly associated with a pelvic abscess, it is a good idea to try rules about them first, before the less likely rules mentioning gram positive rods. Note that this meta-rule does not refer to specific object-level rules. Instead, it specifies certain attributes of the rules it refers to, for example, that they mention in their premise Enterobacteriaceae.

An Example: TEIRESIAS in the Context of MYCIN

We now illustrate TEIRESIAS's operation in affiliation with the MYCIN system (see Article VIII.B1), paying particular attention to the explanation and knowledge-acquisition facilities of TEIRESIAS. MYCIN is intended to provide a physician with advice about the diagnosis and drug therapy for bacterial infections. The user interacts with TEIRESIAS, which in turn communicates with the MYCIN system, although the user is unaware of more than one program being involved. The system asks questions about the patient, the infection, the cultures grown from specimens from the patient, and any organisms

B

(bacteria) growing in the culture. (Typically, of course, the exact identity of the organism is not yet known.)

MYCIN's knowledge base is composed of rules that specify a situation (involving information about the patient, culture, and organism) and the conclusions that can be drawn in that situation. For example, to conclude whether a patient suffers from a bacterium in the Enterobacteriaceae category, MYCIN invokes rule 95:

```
                          RULE 095
IF      The site of the culture is blood, and
        the gram stain is positive, and
        the portal of entry is gastrointestinal tract, and
        [A-the abdomen is the locus of infection, or
         B-the pelvis is the locus of infection]

THEN    There is strongly suggestive evidence that
        Enterobacteriaceae is the class of organisms
        for which therapy should cover.
```

MYCIN asks questions of the physician to determine which rules are applicable in the current case—that is, to determine the "situation." To determine if the organism is of type Enterobacteriaceae, MYCIN would inquire whether each clause in the IF-part of rule 95 were true.

The procedure of working backward from the conclusion to the situation that would make the conclusion true is called *backward chaining* and is MYCIN's fundamental reasoning mechanism. Backward chaining can go several levels deep, meaning that the clauses in a situation that must be true (for the conclusion to be true) might themselves be conclusions of other rules. For example, a condition on the identification of Enterobacteriaceae is that the infection be localized in the abdomen or the pelvis; but this condition is itself a conclusion, and it is only true if its conditions (that the culture from these places is recent and is therapeutically significant) are true. Furthermore, the condition of therapeutic significance has its own conditions, and so on. Here is a protocol showing MYCIN's basic style of interaction (the physician's responses are prefixed by **):

```
———Patient 281———

1) Patient's name:

** Albert Hall

2) Sex:

** Male
```

```
3) Age:

** 34

4) Have you been able to obtain positive microbiological
   information about a possible infection of Albert Hall?

** Yes

--------Infection 1--------

5) What is the infection?

** Primary-Bacteremia

6) Please state when signs and symptoms of infection-1
   first appeared:

** 15-Dec-80 12:00

--------Culture 1--------

7) From what site was the specimen for culture-1 taken?

** Blood

8) Please give the date and time when this blood culture was
   obtained:

** 16-Dec-80 12:00

The first significant organism from this blood culture will be
referred to as:

--------Organism-1--------

9) Enter the identity of organism-1

** Unknown

10) Is organism-1 a rod, or coccus (etc.):

** Rod

11) The gram-stain of organism-1:

** Gram negative

12) Did organism-1 grow in the aerobic bottle?

** Yes

13) Did organism-1 grow in the anaerobic bottle?

** Yes

            :
```

B

```
My therapy recommendations will be based on the following
probable infection(s) and potential causative organism(s):

INFECTION-1 is PRIMARY-BACTEREMIA
    <Item-1> E.COLI [ORGANISM-1]
    <Item-2> PSEUDOMONAS-AERUGINOSA [ORGANISM-1]
```

In sum, MYCIN constructs a *goal tree* (see Article II.B2, in Vol. I) of questions that must be resolved in the course of the consultation to conclude the identity of a bacterium. When it cannot resolve a question by inference from what it already knows, it asks the respondent to provide an answer. Each node of the goal tree has subnodes that result from the application of a rule, as shown in Figure B–1.

Explanation: HOW and WHY. Suppose that during a consultation with MYCIN the clinician wants an explanation of the reasoning behind some question that MYCIN has asked. The explanation given by TEIRESIAS in response to the physician's question, WHY, involves backing up the goal tree to the previous node and presenting the rule used to derive the subnode from its ancestor. (Again, the physician's responses are prefixed by **.)

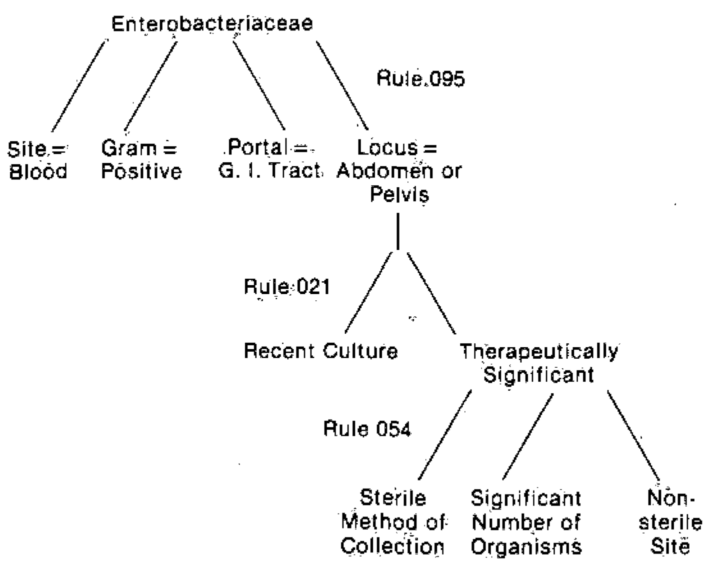


Figure B–1. A MYCIN goal tree.

First, MYCIN's question:

```
What is the suspected portal of entry of organism-1 into this
sterile site?
```

The physician requests an explanation.

```
** WHY
```

Then TEIRESIAS proceeds to explain it's "reasoning."

```
[I.e., why is it important to determine the suspected portal
of entry of organism-1 into this sterile site?]

[3.0] ... in order to find out about an organism or class of
      organisms for which therapy should cover.

It has already been established that
   [3.1] The site of the culture is blood
   [3.2] The stain of the organism is gram positive

Therefore if:
   [3.3] The portal of entry of the organism is G.I.
   [3.4] [Abdomen is the locus of infection, or
           Pelvis is the locus of infection]

Then
   There is strongly suggestive evidence (.9) that
   Enterobacteriaceae is the class of organisms
   for which therapy should cover.

[Rule 095]
```

In contrast to WHY, which spells out "higher" nodes in the goal tree, the HOW command asks TEIRESIAS to explain the nodes lower down the tree that have been or will be expanded. For example, in the following exchange, the expert asks how MYCIN will determine whether the abdomen or the pelvis is the locus of infection:

```
** HOW [3.4]
```

Note: [3.4] is the prefix of the statement of interest in the exchange above.

```
[I.e., how were you determining that the Abdomen is the locus
of infection, or that the Pelvis is the locus of infection?]

[4.0] At that point Rule 021 was being used.
```

If the doctor now wants to see rule 21, he asks TEIRESIAS to explain HOW it was used, referring to the prefix [4.0] of the statement in which it was mentioned:

```
** HOW [4.0]

[I.e., how was Rule 021 used?]

   [5.0] It has already been established that
   [5.1] the culture is recent.

Therefore if:
   [5.2] There is therapeutically significant disease
         associated with the occurrence of this organism

Then
   It is definite (1.0) that the site of the culture
   is the locus of infection in the patient.
```

The explanation facilities of TEIRESIAS involve climbing up and down MYCIN's goal tree, explaining why questions are asked and how they are answered. If MYCIN makes a misdiagnosis, these same facilities are used to back up through the rules that achieved the diagnosis until the expert has found the faulty one or determined that one is missing. TEIRESIAS is then used to expedite the alteration or construction of rules by the expert. (See Article IX.C8 for a discussion of how a similar explanation facility is used in the tutorial system EXCHECK.)

Knowledge acquisition. TEIRESIAS can help an expert formulate rules in English because it has expectations about what rules "look like." These expectations are stored internally as *rule models* that are formed automatically by examining all the rules in the knowledge base. TEIRESIAS knows, for example, that there is a class of rules that have as their conclusions the probable identity of a bacterium and that these rules typically have as conditions (a) the site of the culture, (b) the stain of the culture, and (c) the probable portal of entry of the bacteria. These rule models help TEIRESIAS translate English versions of new rules into a machine representation acceptable to MYCIN.

To illustrate TEIRESIAS's debugging and correction facilities, we will continue with the consultation shown earlier. Suppose that after MYCIN presents the expert with its diagnosis, TEIRESIAS steps in and checks whether the expert agrees (the physician's response is indicated by **):

```
My therapy recommendation will be based on the following
probable infection(s) and potential causative organism(s):

INFECTION-1 is PRIMARY BACTEREMIA
   <Item-1> E.COLI [ORGANISM-1]
   <Item-2> PSEUDOMONAS-AERUGINOSA [ORGANISM-1]
```

Now TEIRESIAS steps in.

```
Are all these identities correct?

** No.

Are there any identities missing?

** Yes.

Which ones?

** KLEBSIELLA-PNEUMONIAE, PROTEUS-NON-MIRABILIS

Are there any identities which shouldn't have appeared?

** Yes.

Which one(s)?

** PSUEDOMONAS-AERUGINOSA

Do you want to
   1] track down the problem yourself, or
   2] allow me to guide the debugging, or
   3] ignore it for now and fix it another time.
Type 1, 2, or 3.

** 2.
```

If the expert had typed 1, he would have used the HOW and WHY commands to debug the problem, as discussed above.

```
We'll start with PSEUDOMONAS-AERUGINOSA first:
```

TEIRESIAS starts debugging by trying to find out why an inappropriate diagnosis occurred. Frequently, solving one problem will make others "go away" as well.

```
The following rules were used in deducing that the identity of
organism-1 is PSEUDOMONAS-AERUGINOSA:

RULE 184
  Since
   [1.1] The category of organism-1 is not known
   [1.2] The gram stain of organism-1 is gram negative
   [1.3] The morphology of organism-1 is rod
   [1.4] the aerobicity of organism-1 is facultative

  There is weakly suggestive evidence (.3) that the identity
  of organism-1 is PSEUDOMONAS-AERUGINOSA.

Is this rule correct?
```

Note that this rule, which resulted in the wrong conclusion, is not necessarily an incorrect rule. It might be (and in fact is) the case that it was incorrectly invoked because one or more of its conditions are true when they should be false.

B

```
** Yes.

Should its premise have failed for this case?

** Yes.

Which clause of the premise should have failed?

** Clause 1.
```

The expert believes that MYCIN should have known the category of ORGANISM-1. MYCIN failed to establish the category, causing rule 184 to be used.

```
Should it have been possible to conclude a category?

** Yes.

Which one?

** Enterobacteriaceae.

There were no rules successfully invoked in the attempt to
deduce that the category of organism-1 is Enterobacteriaceae.

The following (potentially applicable) rules were NOT executed
because the marked [++] clause failed:

RULE 116
  If
   1) The identity of organism-1 is not known
++ 2) The gram stain of organism-1 is not known
   3) The morphology of organism-1 is not known
   4) The site of culture-1 is CSF
   5) The infection is meningitis
   6) The age of Albert Hall is less than or
      equal to 2 months

  Then there is weakly suggestive evidence (.33) that the
  category of organism-1 is Enterobacteriaceae

RULE 050
  If
   1) The morphology of organism-1 is rod
   2) The gram stain of organism-1 is gramneg
   3) The aerobicity of organism-1 is facul
++ 4) The infection with organism-1 was acquired
      while the patient was hospitalized

  Then there is suggestive evidence (.7) that the category of
  organism-1 is Enterobacteriaceae.

Are all these rules correct?

** Yes.
```

```
Should any of them have been successfully invoked?

** No.
```

This would be the case if one of the marked premises was false because it was "stepped on" by yet another rule farther back in the chain.

```
Is there a rule missing?

** Yes.

Would you like to give me such a rule now?

** Yes.

Good ...
```

The expert then proposes another rule to TEIRESIAS:

```
If:
   1 ** THE PATIENT'S INFECTION IS PRIMARY-BACTEREMIA
   2 ** THE SITE IS ONE OF THE STERILE SITES

Then:
     ** THERE IS EVIDENCE (.8) THAT THE CATEGORY IS
        ENTEROBACTERIACEAE
```

TEIRESIAS parses this rule into its internal representation and then translates it back into English, so that the expert can check it. We pick up the dialogue after the expert has agreed on the parse. TEIRESIAS is applying its rule model.

```
I hate to criticize, Dr. Davis, but did you know that most
rules about what the category of an organism might be,
that mention
   the site of a culture
   the infection

Also mention -
   [A] the portal of entry of the organism

Shall I try to write a clause to account for [A]?

** Yes.
```

Here we see one of the utilities of the rule model. TEIRESIAS prompts the expert to add a bit of knowledge that is typically found in rules like these and offers to fix the rule itself. This completion is accomplished by looking at other rules that fit the same rule model, to find the most likely portal-of-entry clause.

```
How about -
   [A] The portal of entry is gastrointestinal.  OK?

** Yes.
```

B

TEIRESIAS then does some finishing up: checking the complete rule with the expert for final approval and asking the expert to write a brief description (for bookkeeping purposes) of why the rule was needed. Finally, it runs the consultation again internally, using the responses from the early part of the dialogue, which it has stored. It turns out that adding the rule above did, in fact, correct the other problems with the first consultation, and this time the diagnosis is satisfactory to the expert.

Summary: TEIRESIAS and Expert Systems

TEIRESIAS aids a human expert in monitoring the performance of a knowledge-based system. When the human expert spots an error in the program's performance, in either the program's conclusions or its line of reasoning, TEIRESIAS assists in finding the source of the error in the database by *explaining* the program's conclusions—retracing the reasoning steps until the faulty (or missing) rule is identified. At this point, TEIRESIAS assists in *knowledge acquisition*, modifying faulty rules or adding new rules to the database. *Meta-level knowledge* about the kinds of rules and concepts in the database is applied to build expectations in TEIRESIAS's *model-based understanding* process. Meta-level knowledge is also used to encode problem-solving *strategies*, in particular, to order the invocation of rules so that those that are most likely to be useful (given the current knowledge of the program) are tried first.

References

The principal reference on TEIRESIAS is the doctoral dissertation by Davis (1976). Applications of meta-knowledge in expert systems are discussed in Davis and Buchanan (1977). See also Davis (1977, 1978, 1980).

D3. PROSPECTOR

PROSPECTOR (Duda et al., 1978) is a computer-based consultation system that is being developed at SRI International to assist geologists working on certain problems in "hard-rock" mineral exploration. Like other expert systems, such as MYCIN (see Article VIII.B1) and INTERNIST (Article VIII.B3), PROSPECTOR attempts to represent a significant portion of the knowledge and the reasoning processes of experts working in a specialized domain. The intended user of this program is an exploration geologist who is in the early stages of investigating an exploration site, or "prospect." We assume that he (or she) has a professional understanding of geology but, nonetheless, wants the assistance of a specialist in evaluating the findings.

In an attempt to keep the PROSPECTOR system domain-independent, a clear separation is maintained between the geological knowledge base and the mechanisms that employ this knowledge (although characteristics of the problem domain have certainly influenced the design of the system). Expressed abstractly, the main function of PROSPECTOR is to match data from a particular situation against *models* that describe a moderately large number of disjoint classes of situations. In PROSPECTOR's domain, the models are formal descriptions of the most important types of ore deposits, and the data are primarily surface geological observations. The available data are assumed to be uncertain and incomplete, so that the conclusion is expressed as a probability or a degree of match. In addition, the program alerts the user to different possible interpretations of the data and identifies additional observations that would be most valuable for reaching a more definite conclusion.

A typical consultation session with PROSPECTOR begins with the user giving the system the information about the most significant features of his prospect: the major rock types, minerals, and alteration products. The program matches these observations against its models and, when the user has finished volunteering information, proceeds to ask the user for additional information that will help confirm the best matching model. At any time during the consultation, the user can interrupt to volunteer new information, change previous statements, or request an evaluation.

The following edited excerpt from a PROSPECTOR session illustrates many of these ideas. The data for the run describe a fictitious porphyry copper deposit that has some, but not all, of the desired characteristics. We begin at the point where the user (indicated by **) starts volunteering information to the system.

```
** There is quartz monzonite

Quartz monzonite (5)
```

```
** There is a cretaceous dike

Cretaceous dike (5)

** There is sericite and pyrite

Sericite (5) pyrite (5)

** There may be biotite

Biotite (2)

** Done
```

These simple assertions describe the most prominent rocks, minerals, and alteration products in the target area. The system indicates its recognition of each statement by echoing the statement and appending an assigned certainty. Certainty factors range from −5 (positive that a situation does not hold) to 5 (positive that it does hold). All assertions made by PROSPECTOR have associated certainty factors. The program now enters the consequent mode of reasoning.

```
Presently, the models in PROSPECTOR and their scores are:
  1. Komatiitic nickel sulfide deposit -- KNSD (0.0)
  2. Type-A porphyry copper deposit -- PCDA (0.0)
     ++current hypothesis++
  3. Kuroko-type massive sulfide deposit -- MSD (.001999)
  4. Western-states-type sandstone uranium deposit -- WSSU (0.0)
  5. Mississippi-Valley-type lead/zinc deposit -- MVTD (-4.8)
Which of the above would you like to rule out?

** None
```

The volunteered information was sufficient to rule out one of the models but was not nearly specific enough to establish any of the others. The choice of the second model as the current hypothesis was based not on its certainty but on the number and types of connections between it and the volunteered statements. To avoid a possibly fruitless investigation of unpromising models, the system gives the user the opportunity to exclude models of lesser interest. In this case, the user allows all of the models to be explored, and the system begins with the current hypothesis.

```
I am considering the possibility of a
( Type-A porphyry copper deposit -- PCDA )
in the target area.

The following questions are intended to establish the nature
of the petrotectonic setting.

1--To what degree do you believe that:
( the target area is in a continental margin mobile belt ) ?

** -2
```

D3

This is a discouraging answer for the PCDA model, and it removes a context needed for the investigation of other properties of the continental-margin mobile belt.

```
The following questions are intended to determine the nature
of the regional environment.

2--To what degree do you believe that:
( there are granitic intrusives in the region ) ?

** 5

3--To what degree do you believe that:
( the region contains an abundance of small stocks ) ?

** -3
                :

8--To what degree do you believe that:
( igneous rocks in the region have porphyritic texture ) ?

** Why

I am trying to establish whether some of the intrusive rocks in
your area have textures suggestive of a hypabyssal to subvolcanic
environment.  Porphyry textures resulting from the rapid cooling
and crystallization of a silicate melt which contains large
mineral grains are suggestive of relatively shallow levels
of emplacement.
```

Like MYCIN, PROSPECTOR provides a variety of commands for such things as requesting a summary, changing previous answers, or volunteering additional information. The above response to a WHY command is not automatically generated from the rule but is merely stored text associated with the space.

```
8---To what degree do you believe that:
( igneous rocks in the region have porphyritic texture ) ?

** 5

My certainty in ( the texture and morphology of the intrusive
system suggest a subvolcanic (hypabyssal) regional environment )
is now: 4.924

Do you want to pursue this further?

** No
```

At this point the system noted that the certainty for the hypothesis about a hypabyssal environment was above 4.5, and it also noted that if the remaining rules were answered in the most unfavorable way possible, the certainty could not fall below 4.5. To save time, it let the user decide whether or not to continue working on this subgoal.

The questioning continues as above until it becomes necessary to consider properties that vary within the target area. Thus, for the first time it becomes necessary to be able to use the rules more than once, and a communication convention must be established to distinguish different spatial locations.

```
Porphyry copper deposits are often composed of characteristic
zones of alteration and mineralization.  (In some cases these
can be looked upon as different stages superimposed upon one
another.)  I shall call the most prominent such zone Zone 1.
Even though you may have already identified this zone as, say,
the propylitic zone or the sericitic zone, I need to consider
other possibilities as well.  Thus, the following questions
are intended to determine the possibilities for Zone 1.

25--To what degree do you believe that:
( there are fresh-looking feldspars in Zone 1 ) ?

** -5
```

Questioning continues in this manner. Eventually, PROSPECTOR establishes that two zones exist in the region of interest and, after examining each in detail, it concludes that their geologic makeup supports the hypothesis that Type-A porphyry copper exists in the region.

```
On a scale from -5 to 5, my certainty that ( there are alteration
zones that are favorable for a Type-A porphyry copper deposit )
is now 4.833

There are two favorable factors; in order of importance:
  2.3.1: 1.  there is a sericitic zone in Zone 1 (4.834)
             ++ establishes 2.3.1 ++
  2.3.1: 2.  there is a potassic zone in Zone 2 (4.499)

There are many factors that would have been unfavorable had no
favorable factor existed to override them; in order of importance:

  2.3.1: 3.  there is a barren-core zone in Zone 1 (-5.0)
  2.3.1: 4.  there is a potassic zone in Zone 1 (-5.0)
  2.3.1: 5.  there is a propylitic zone in Zone 1 (-5.0)
  2.3.1: 6.  there is a sericitic zone in Zone 2 (-5.0)
  2.3.1: 7.  there is a propylitic zone in Zone 2 (-4.989)
  2.3.1: 8.  there is a barren-core zone in Zone 2 (-4.495)

For which of the above do you wish to see additional information?
                    :
```

Knowledge Representation

The data structure for representing the geological knowledge embodied in PROSPECTOR is called the *inference network* and guides the plausible reasoning performed by the system. The nodes in this network correspond to various assertions, such as *There is pervasively biotized hornblende* or *There* is alteration favorable for the potassic zone of a porphyry copper deposit. In a particular run, any assertion may be known to be true, known to be false, or suspected to be true with some probability.

Most of the arcs in the inference network define *inference rules* that specify how the probability of one assertion affects the probability of another assertion. For example, the presence of pervasively biotized hornblende suggests the potassic zone of a porphyry copper deposit, and the absence of any biotized hornblende is very discouraging for that conclusion. These inference rules correspond to the production rules used in MYCIN. The remaining arcs indicate that an assertion is the context for another assertion, preventing conclusions from being drawn until the right contexts are established. For example, one should establish that hornblende has been altered to biotite before asking about the degree of alteration.

The primary task confronting a geologist who wants to prepare a new model for PROSPECTOR is the representation of his model as an inference network. The current system contains models of five different types of deposits, developed in cooperation with five different consulting geologists. The statistics in Table D3-1 give a rough indication of the size and complexity of these models.

To allow certain kinds of logical reasoning by the system, each assertion is represented as a *space* in a *partitioned semantic network* (see Article III.C3, in Vol. I). A typical space asserts the hypothetical existence of physical entities having specific properties (such as being composed of biotite) and participating in specific relations (such as an alteration relation). In addition, a large taxonomic network describes important element-subset relations among the terms mentioned, such as the fact that biotite is a mica, which in turn is a silicate, which in turn is a mineral.

The articulation of assertions as a set of relations allows the system to recognize subset-superset connections between pairs of assertions. For example, the assertion *There is pervasively biotized hornblende* is clearly related to the assertion *There is mica;* assertion of the first also asserts the second,

TABLE D3-1

Size of Knowledge Base of Five PROSPECTOR Models

Model	Number of assertions	Number of rules
Koroko-type massive sulfide	39	34
Mississippi-Valley-type lead/zinc	28	20
Type A porphyry copper	187	91
Komatiitic nickel sulfide	75	49
Roll-front sandstone uranium	212	133
Total	541	327

and denial of the second denies the first. This kind of recognition is used in two main ways. First, it provides important intermodel and intramodel connections beyond those given explicitly by the inference rules. Second, it allows the system to recognize connections between information volunteered by the user and the coded models.

Probabilistic Reasoning

Some of the logical constraints that hold between spaces have probabilistic implications. In particular, if A is an instance (i.e., subset) of B, then the probability of A can never exceed the probability of B. We maintain this constraint by automatically generating certain inference rules. For example, if evidence E could raise the probability of A above the probability of B, we generate a rule from E to B that will increase the probability of B sufficiently to just satisfy the constraint. The exact procedure used here is described in Duda et al. (1977).

The various inference rules connect to form an inference network; thus, when the user provides some evidence, this information can change the probabilities of several hypotheses, which in turn can change the probabilities of hypotheses that depend on them. The probability formulas determine exactly how these probability changes propagate through the inference net. (The reader might also refer to Articles VIII.B2 and VIII.B6, on CASNET and IRIS, for alternative methods of propagation.)

Control

PROSPECTOR is a mixed-initiative system that begins by allowing the user to volunteer information about the prospect. This volunteered information is currently limited to simple statements in constrained English about the names, ages, and forms of the rocks and the types of minerals present. These statements are parsed by LIFER, a natural-language interface facility (see Article IV.F7, in Vol. I), and represented as partitioned semantic networks. A network-matching program compares each of these volunteered spaces against the spaces in the models, noting any subset, superset, or equality relations that occur.

If a volunteered space is exactly equal to a space in a model, the probability of the model space is updated and that change is propagated through the inference network. If a volunteered space is a subset of a space in a model and if it has a higher probability than the model space, once again the probability of the model space is updated and that change is propagated through the inference network.

Unfortunately, if the volunteered space matches a superset of a model space (which is usually the case), no probability change can be made unless the user expresses doubt about the situation. For example, if the user mentions biotite, the probability of the space that asserts that there is pervasively biotized hornblende is unchanged, unless the user has said that he doubts that there is any biotite. However, it is obvious that the system may want to follow up this observation, and the existence of the connection to the model is recorded.

When the user has finished the initial volunteering, PROSPECTOR scores the various models on the basis of the number and types of connections that have occurred and selects the best matching model for further investigation. Here, the basic control strategy is MYCIN-like backward chaining or consequent reasoning. At any given time, there is a current goal space whose existence is to be determined. The initial goal space is the one that corresponds to the best matching model. The various spaces in the models represent either evidence that can be sought from the user (are "askable") or internal hypotheses that are to be deduced from evidence (are "unaskable"). Naturally, the initial goal space is always unaskable. If the current goal space has any unestablished context spaces, they are pushed on the goal stack and one of them becomes the new current goal.

If the current goal is askable and has not been asked before, the user is asked about it, the effects of the answer are propagated through the inference network, and the process is repeated. If it is unaskable, it must be either the consequence of one or more inference rules or a logical combination of one or more other spaces. In the former case, the rules are scored to determine their potential effectiveness in influencing H, and the antecedent of the best scoring rule becomes the next goal. In the latter case, a predetermined supporting space becomes the next goal. In either case, the same procedure is repeated until (a) the top-level goal becomes so unlikely that another top-level goal is selected, (b) all of the askable spaces have been asked, or (c) the user interrupts with new volunteered information.

Summary

This brief overview covers the basic knowledge-representation and inference mechanisms used in PROSPECTOR. Many aspects of the system have not been discussed, such as the treatment of quantitative evidence, the matching procedure, the use of graphical input, the inference-network compiler, the explanation system, model-acquisition aids, and the test and evaluation effort.

The five models in the current system are but a fraction of what is needed for comprehensive coverage of the prospecting domain, and even these models have only recently reached the degree of completeness required for doing meaningful evaluations. Limited initial tests have shown very close agreement between the evaluations provided by the system and the evaluations of the model designers, using data from actual deposits of the types modeled. And, in fact, PROSPECTOR recently made a prediction about the location of molybdenum ore at an exploration site in the state of Washington that was substantially confirmed by subsequent drilling. More information on the system, the extent of its geological knowledge, its performance on known deposits, and its possible applications can be found in Duda et al. (1978).

References

Duda, Gaschnig, and Hart (1979) is a brief description of PROSPECTOR. See also Duda et al. (1977, 1978).

On the Development of Commercial Expert Systems

Reid G. Smith
Schlumberger-Doll Research
Old Quarry Road
Ridgefield, Connecticut 06877

Abstract

We use our experience with the Dipmeter Advisor system for well-log interpretation as a case study to examine the development of commercial expert systems. We discuss the nature of these systems as we see them in the coming decade, characteristics of the evolution process, development methods, and skills required in the development team. We argue that the tools and ideas of rapid prototyping and successive refinement accelerate the development process. We note that different types of people are required at different stages of expert system development: Those who are primarily knowledgeable in the domain, but who can use the framework to expand the domain knowledge; and those who can actually design and build expert system tools and components. We also note that traditional programming skills continue to be required in the development of commercial expert systems. Finally, we discuss the problem of technology transfer and compare our experience with some of the traditional wisdom of expert system development.

THE PAST DECADE has seen the development of a number of expert systems, mostly by AI researchers for use in research environments. To date, few have been utilized for industrial applications. As a result, we have little experience with which to characterize either the nature of commercial expert systems or their development process.

The Dipmeter Advisor system is the result of a four year effort by Schlumberger to apply expert systems technology to problems of well-log interpretation. We have observed during this effort that the development of a commercial expert system imposes a substantially different set of constraints and requirements in terms of characteristics and methods of development than those seen in the research environment.

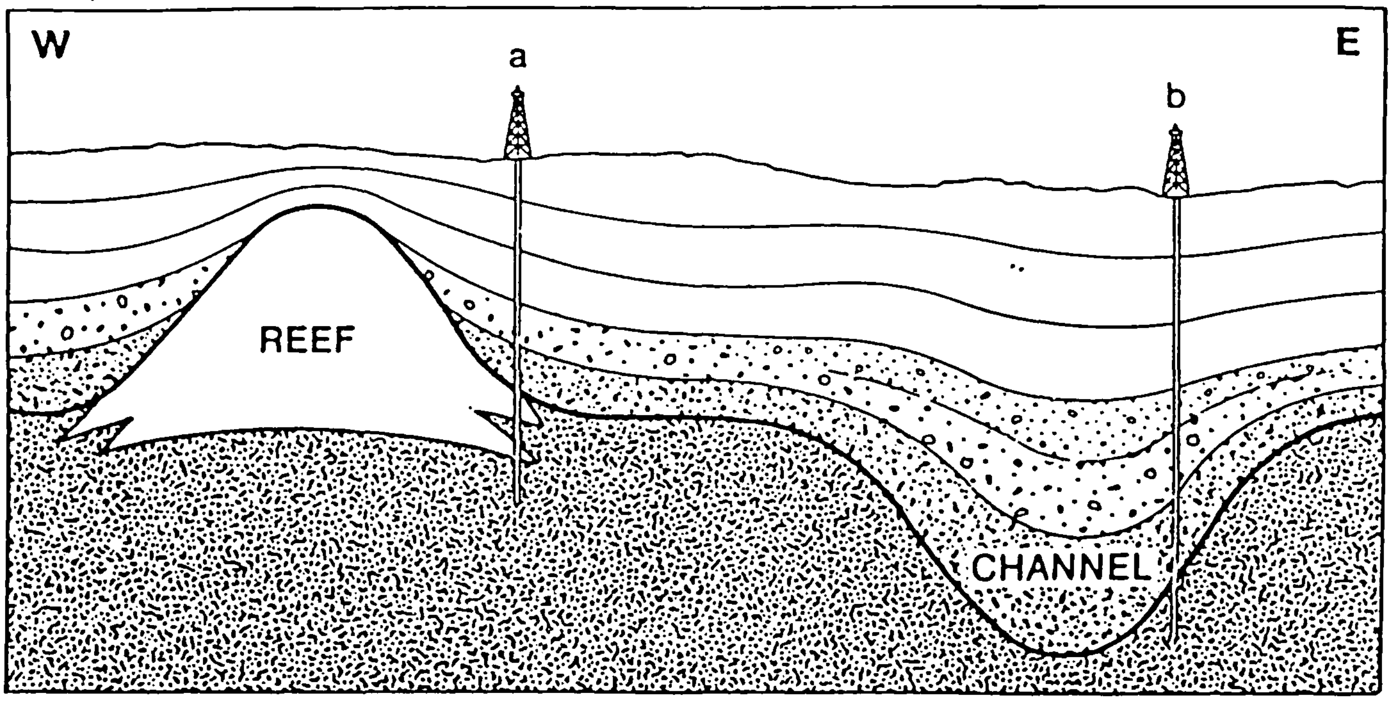
This article is intended as a case study. We briefly describe the dipmeter interpretation problem and the evolution of the Dipmeter Advisor system. During its development a number of ideas have surfaced that we believe to be characteristic of this type of effort, given the current state of the technology. While the data are too sparse for definitive results, these ideas are thought to be important and suggestive as guidelines for subsequent commercial expert system undertakings.

Example: Dipmeter Interpretation

The Problem

Oil-well logs are made by lowering tools into the borehole and recording measurements made by the tools as they are raised to the surface. The resulting logs are sequences of values indexed by depth. Logging tools measure a variety of petrophysical properties. The dipmeter tool in particular

David Barstow, J. A. Gilreath, Tom Mitchell, and Peter Will made a number of helpful suggestions for this paper. David Gallo and Chip Hendrickson provided the football figures.



Dip in Subsurface Formations.

Figure 1.

measures the conductivity of rock in a number of directions around the borehole. Variations in conductivity can be correlated and combined with measurements of the inclination and orientation of the tool to estimate the magnitude and azimuth of the dip or tilt of various formation layers penetrated by the borehole (Figure 1).

Because the dipmeter tool has high resolution in the vertical direction (0.1-0.2 in.), it provides the petroleum geologist with detailed information on relatively fine-structured sedimentary beds. This type of information is invaluable in defining hydrocarbon reservoir structure and designing methods to drain such reservoirs.

Knowledge of the dip variations as a function of depth in the vicinity of the borehole does not in itself identify geologic features. However, when combined with knowledge of local geology and rock properties measured by other logs (*e.g.*, lithology (sand, shale,)), the characteristic dip patterns (signatures) of geologic events in the depositional sequence can be interpreted.

The right channel of Figure 2 is an interval of a dipmeter log. Dip estimates are shown as tadpoles. Dip magnitude increases to the right of the graph, and the down dip direction is indicated by the tail on each tadpole. The vertical axis is depth. Hollow tadpoles indicate lower confidence dip estimates than solid tadpoles. (So, for example, the tadpole at 8360 ft. indicates a formation that is dipping down to the southeast at approximately 24°.) The left channel is a gamma ray log. (It measures natural gamma radiation in the formation—a rudimentary lithology indicator.)

Sequences of tadpoles can be grouped together in patterns. Three of the characteristic dip patterns are described below (Schlumberger, 1981).

- *Green Pattern*: An interval (zone) of constant dip magnitude and azimuth. This pattern is characteristic of structural dip—caused by large-scale tectonic disturbance that occurs long after deposition and compaction of sediment.
- *Red Pattern*: A zone of increasing dip magnitude with constant azimuth over depth. This pattern is indicative of down dip thickening, which may be associated with distortions near structural features (*e.g.*, faults), differential compaction of sediment over buried topographic features (*e.g.*, reefs), or channel filling.
- *Blue Pattern*: A zone of decreasing dip magnitude with constant azimuth over depth. This pattern is indicative of down dip thinning, which may be associated with distortions near structural features, differential compaction beneath denser overlying deposits (*e.g.*, sand lenses), or sediment transport by water or wind.

From this localized data, a skilled interpreter is often able to make comprehensive deductions about the geological history of deposition, the composition and structure of the beds, and the optimum locations for future wells.

The Dipmeter Advisor System

The Dipmeter Advisor system attempts to emulate human expert performance in dipmeter interpretation. It utilizes dipmeter patterns together with local geological knowledge and measurements from other logs. It is characteristic of the class of programs that deal with what has come to be known as signal to symbol transformation (Nii, 1982). The program is written in INTERLISP-D and operates on the Xerox 1100, 1108, or 1132 Scientific Information Processor.[1]

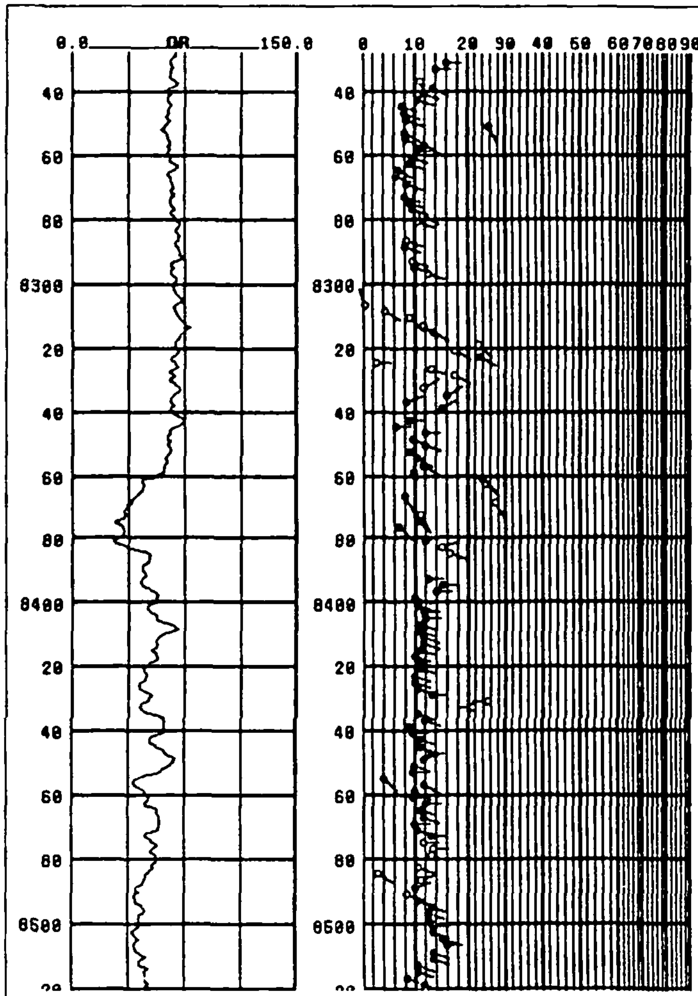
The system consists of four central components: a number of production rules partitioned into several distinct sets according to function (*e.g.*, structural rules vs stratigraphic rules); an inference engine that applies rules in a forward-chained manner, resolving conflicts by rule order; a set of feature detection algorithms that examines both dipmeter and open-hole data (*e.g.*, to detect tadpole patterns and identify lithological zones); and a menu-driven graphical user interface that provides smooth scrolling of log data.

Conclusions are stored as instances of one of 65 token types, with approximately 5 features/token, on a blackboard that is partitioned into 15 layers of abstraction (*e.g.*, patterns, lithology, stratigraphic features). There are 90 rules, and the rule language uses approximately 30 predicates and functions. The rules have the empirical association flavor. A sample is shown below.[2]

```
IF
        there exists a delta-dominated continental-shelf
        marine zone, and
        there exists a sand zone intersecting the marine
        zone, and
        there exists a blue pattern within the intersection.
THEN
        assert a distributary fan zone
           top ← top of blue pattern
           bottom ← bottom of blue pattern
           flow ← azimuth of blue pattern
```

The system divides the task of dipmeter interpretation into eleven successive phases as shown below. After the system completes its analysis for a phase, it engages the human interpreter in an interactive dialogue. He can examine, delete, or modify conclusions reached by the system. He can also add his own conclusions. In addition, he can revert to earlier phases of the analysis to refer to the conclusions, or to rerun the computation.

- **Initial Examination:** The human interpreter can peruse the available data and select logs for display.
- **Validity Check:** The system examines the logs for evidence of tool malfunction or incorrect processing.
- **Green Pattern Detection:** The system identifies zones in which the tadpoles have similar magnitude and azimuth.



Gamma Ray and Dipmeter Logs

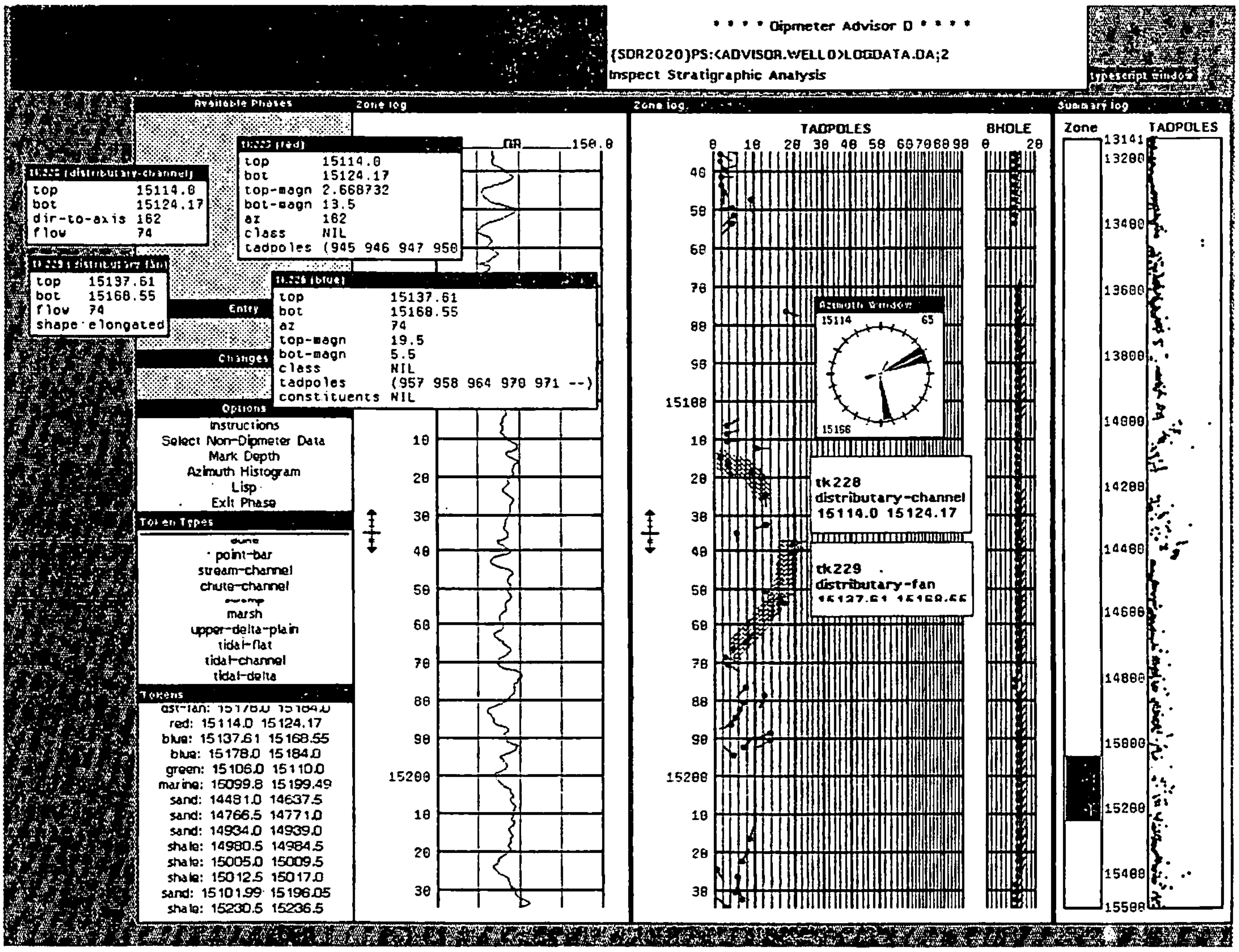
Figure 2.

- **Structural Dip Analysis:** The system merges and filters green patterns to determine zones of constant structural dip.
- **Preliminary Structural Analysis:** The system applies a set of rules to identify structural features (*e.g.*, faults).
- **Structural Pattern Detection:** The system examines the dipmeter data for red and blue patterns in the vicinity of structural features.[3]
- **Final Structural Analysis:** The system applies a set of rules that combines information from previous phases to refine its conclusions about structural features (*e.g.*, strike of faults).
- **Lithology Analysis:** The system examines the open hole data (*e.g.*, gamma ray) to determine zones of constant lithology (*e.g.*, sand and shale).
- **Depositional Environment Analysis:** The system applies a set of rules that draws conclusions

[1] Early versions of the program are described in (Davis, 1981) and (Gershman, 1982).

[2] This sample is similar to the actual interpretation rule, but has been simplified somewhat for presentation.

[3] The algorithms used by the system to detect dip patterns are beyond the scope of this paper.



Stratigraphic analysis interaction

Figure 3.

about the depositional environment. For example, if told by the human interpreter that the depositional environment is marine, the system attempts to infer the water depth at the time of deposition.

- **Stratigraphic Pattern Detection:** The system examines the dipmeter data for red, blue, and green patterns in zones of known depositional environment.
- **Stratigraphic Analysis:** The system applies a set of rules that use information from previous phases to draw conclusions about stratigraphic features (*e.g.*, channels, fans, bars).

For the phases shown above, "+" indicates that the phase uses production rules written on the basis of interactions with an expert interpreter. The remaining phases do not use rules.[4]

Figure 3 shows a sample Xerox 1100 screen following the stratigraphic analysis phase. On the extreme right the system displays a summary log of dip magnitude for the entire well. The black box indicates the region of the well that is expanded in the second window from the right. This window shows the dipmeter data together with the deviation of the borehole itself. The next window displays two other logs: GR (gamma ray) and ILD (a resistivity log). (Each of these windows can be smoothly scrolled by moving the mouse into its speed bar, one of which is visible on the left side of the dipmeter window. A more radical movement can be achieved by moving the mouse into the black elevator box visible on the right side of the dipmeter window. The size of the interval viewable in the dipmeter and other log windows is also under mouse button control.)

The system summarizes relevant conclusions in the (scrolling) windows in the lower left hand part of the screen. The

[4]The rules obtained to date are due to J. A. Gilreath of Schlumberger Offshore Services, New Orleans, LA. The feature detectors and signal-processing algorithms were written independently by project members.

user (a dipmeter interpreter) has selected a number of conclusions to be examined in greater detail and shown as annotations on the dipmeter log. Also shown is the dip azimuth trend before and after structural dip removal.[5]

Building Commercial Expert Systems

Embedded Systems

Domain practitioners are typically much more interested in the utility and performance of a system that is to help them solve their problems than in the particular methods used to construct it. Furthermore it is unlikely that traditional AI methods alone will solve real problems. They are likely to be augmented by techniques from signal processing and pattern recognition, to name but two possibilities. This implies that the computer scientist involved in commercial expert system development must be prepared to solve problems that involve a variety of disciplines and techniques.

It is our view that the expert system kernel is likely to be a (perhaps even relatively small) component embedded in a larger system. The particular suite of problems common to signal understanding problems may, of course, bias our outlook, but we believe that it is difficult to avoid the conclusion that acceptance and real use of expert systems depend on far more than a knowledge base and inference engine.[6]

Indeed our experience has been that these traditional parts of an expert system are not the predominant parts of the overall system either in terms of the amount of code or the resources required for system development. It is instructive in this regard to examine the relative amounts of code devoted to various functions in the Dipmeter Advisor system:

Inference Engine:	8%
Knowledge Base:	22%
Feature Detection:	13%
User Interface:	42%
Support Environment:	15%

This breakdown cannot be used, of course, as a direct measure of programming effort or as an indicator of where the system gets its power. However, the human interface figure especially is familiar to designers of expert systems like MYCIN (Shortliffe, 1976), (VanMelle, 1981) and PROSPECTOR/KAS (Reboh, 1981). It demonstrates the importance of a good programming language, and indicates that traditional programming skills continue to be required for the development of commercial expert systems.

[5]The scrolling graphics code was written by Paul Barth and Tony Passera. Extensions to the INTERLISP-D menu package were written by Eric Schoen.

[6]Gaschnig has made a similar observation in the context of the PROSPECTOR system (Gaschnig, 1982).

System Evolution

Based on our experience, we hypothesize an oscillating focus of attention in commercial expert system development projects. Initially, the focus is a demonstration of feasibility; acquiring the knowledge for a constrained problem and finding the appropriate set of expert system tools with which to encode and apply the knowledge. This phase could be relatively short. It is followed by a phase of expansion of the domain knowledge—during which the expert system tools remain relatively constant. A point will likely come at which the intial tools do not provide sufficient power to allow continued expansion of the system's expertise. At that point, the focus will move away from domain problems and toward selection—more likely development—of new expert system tools. Once a new set of more powerful tools has been constructed, then the focus will again return to the domain problems at hand.

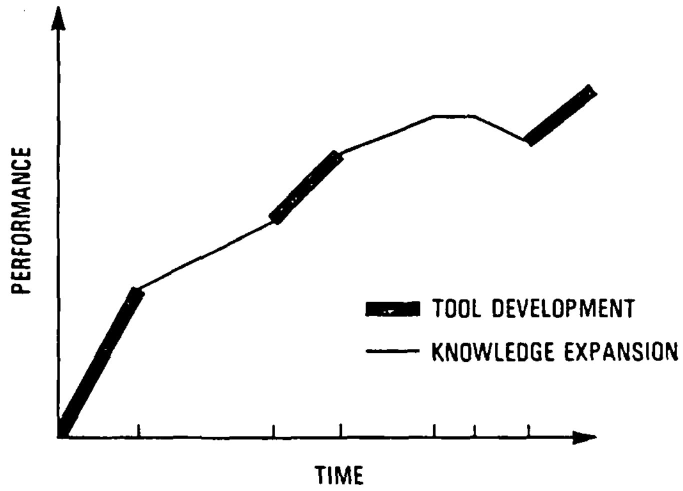
Naturally any particular system may not pass through very many of these oscillations. The focus in the R1 project, for example, didn't appear to oscillate at all (McDermott, 1981). We believe this is due to the nature of the task. There was little of the uncertainty about the nature of the problem that is evident in the the signal understanding or diagnosis tasks. Consequently the initial tools were in fact sufficiently powerful to handle the problem.

In the MYCIN project we seem to be observing the beginnings of an oscillation. The initial system was constructed. Then the rule base was expanded, leaving the initial expert system tools intact. More recently a new design, NEOMYCIN, has appeared—a new set of tools (Clancey, 1981).

Along with the oscillating focus, we hypothesize a rough performance versus time curve. For this discussion performance is taken to include factors such as computation time, accuracy of solutions, and breadth of coverage. We expect this curve to show periods of high positive slope corresponding to implementation of new expert system tools, followed by periods of lower slope corresponding to expansion of domain knowledge, followed by periods of level or even decreasing slope corresponding to reaching (or surpassing) the amount of domain knowledge and generality that can be supported by the tools.

Figure 4 shows the type of performance improvement that we hypothesize, together with the relative emphasis. The emboldened portions of the graph indicate periods of new expert system tool development. The remaining portions correspond to periods of expansion and refinement of domain knowledge. (During the startup period, of course, the two activities proceed concurrently.)

It is currently the case that the precise set of tools required to solve a given problem cannot be accurately predicted *a priori*. Periods of domain knowledge expansion using relatively stable tools are required to expose problem areas and focus tool selection and development. As experience with expert systems grows, for any given problem,



Expert System Incremental Development

Figure 4.

development will effectively begin further along the curve. Developers will start with a better understanding of the set of tools that will eventually be required to solve the problem.

At any given point in time, then, an expert system may require improvement in terms of domain knowledge and expert system tools. Just as the focus of the project will vary, depending on which of the two types of improvement is most pressing, the type of person required to improve the system will also vary.

Improvements in the first area can be made to a large extent by people primarily knowledgeable in the domain, but not necessarily knowledgeable in the design of expert systems. For example, at one stage of its development the Dipmeter Advisor system was familiar with a relatively small number of different lithologies. The performance of the system could be improved in this area without redesign. Similarly, the coverage of the rules could be extended to handle more environments, or specialized to handle local anomalies.[7]

Improvements in expert system tools cannot be made without redesign. This type of effort requires a person who can build such systems, as opposed to one who can use the framework to expand capabilities. For example, the Dipmeter Advisor system uses rule order to help circumvent potential multiple interpretations for the same interval in the well, or simply draws multiple conclusions for the same zone. The human interpreter must select the correct interpretation. The system also has a very local view of consistency in the vertical sequence. This is attributable to the fact that it is reasoning from sets of empirical rules and has no model of the underlying geological processes that lead to the rules. Improvements in these areas cannot be made without redesign.

[7] We have already noted, however, the likelihood that traditional programming skills will continue to be required.

System Development

We have attempted a critical review of the development side of the Dipmeter Advisor system. Although we are as yet unable to abstract a development methodology, several observations stand out. Almost every major issue and decision in the evolution of the Dipmeter Advisor system addressed one or more of the following:

- Demonstration of Feasibility
- Demonstration of Utility and Performance
- Evaluation of Utility and Performance.

Demonstration of Feasibility: The problem of dipmeter interpretation was initially selected as a vehicle for investigating the applicability of expert system techniques to well-log interpretation. Until feasibility could be demonstrated, other questions were secondary.

As a first step, a substantial effort was expended on acquisition of dipmeter interpretation knowledge. This effort was carried out over a 12 to 18 month period using standard techniques (protocols, videotape, discussion, representative examples, and so on). A single expert was studied in detail, again adhering to standard practice.

The implementation of a prototype system followed data acquisition and was carried out in approximately four months (completed in December 1980). The rule base and inference engine were written in INTERLISP (245 Kbytes of source code) and ran on a DEC 2020. The user interface was graphical, written in FORTRAN (450 Kbytes of source code), and ran on a RAMTEK 9400 connected to a VAX 11/780. The VAX and 2020 were linked via a CHAOSnet. The rule base was made up of aproximately 30 rules. There were also several feature detectors and signal processing algorithms.

Demonstration of Utility and Performance: The prototype system demonstrated to the expert that significant analyses were possible. To determine commercial viability, other issues must be addressed. Does the system solve enough of the problem to be interesting and useful? Can the system perform with the efficiency and interactivity necessary in a field environment without overutilizing available computing resources?

Two examples demonstrate the problem. The initial prototype had no means of actually detecting the red and blue patterns and the lithology zones that are required to perform an unaided interpretation. It did not solve enough of the problem to be useful. This lack resulted in implementation of algorithms for simple detection of tadpole patterns and lithologic zones.

Second, the detection of green patterns and determination of structural dip took approximately 18 minutes in the first test well. This duration was unacceptable for actual use—later effort reduced the time to under 2 minutes.

Evaluation of Utility and Performance: Field evaluation was the next hurdle for the Dipmeter Advisor system. Several questions had to be addressed. First, was the rule base sufficiently complete to solve correctly a wide variety of problems in the geological environments for which it was de-

Several questions had to be addressed. First, was the rule base sufficiently complete to solve correctly a wide variety of problems in the geological environments for which it was developed? Second, what changes and effort would be required when working in other geological environments? And third, did the rule base sufficiently capture the thinking of enough dipmeter interpreters to be useful?

To date, this has been the most difficult area. People in the engineering and field groups had to address the above questions. To accomplish this, the prototype system had to be capable of operating in their existing environment—possibly upgraded with modest investment.

One of the difficulties with the initial prototype was the unusual architecture of linked computers, which was not a standard company configuration. In an effort to facilitate testing, the system was reimplemented in FRANZLISP (except for the graphical interface), totally on the VAX 11/780. Unfortunately this change did not solve the problem. The VAX/RAMTEK configuration, as a shared resource in a generally overloaded situation, required an excessively long time to complete a case. Under worst conditions, it took several hours. (In an unloaded VAX environment, it could be completed in one-half hour or less.)

At this point, new technology came to the rescue, and the system was re-implemented on the Xerox 1100, which has both a dedicated processor and sophisticated graphics. In this implementation the graphical interface code was integrated into the remainder of the system. The result was approximately 612 Kbytes of INTERLISP-D source code. This implementation was robust enough and fast enough to allow transfer to a Schlumberger Interpretation Engineering group for testing in a non-research environment.

We can summarize this section as follows: A commercial expert system is ultimately constructed to solve a real problem (as opposed to being constructed, say, to determine the limits of a problem-solving architecture). As a result, the developers should avoid a demonstration mentality. Careful thought at all stages of development about the eventual disposition of the system may prevent the necessity for multiple re-implementations.

The Development Team

Development of a commercial expert system requires people with a variety of skills. The following set is typical. We will expand on it in the remainder of this section.

- Domain Expertise
- Knowledge Engineering
- Expert System Tool Design
- Programming Support

First, it goes without saying that committment of one or more articulate domain experts is crucial to the success of any expert system development.

The term knowledge engineer is normally used to mean computer scientist intermediary—the link between expert and machine. We have divided this role into two parts in order to emphasize that two different kinds of task are involved. The development team normally requires at least one member to interact with the domain experts and encode domain knowledge. We reemphasize that the interaction and encoding activities do not necessarily require someone who can construct expert systems, but rather someone who can become knowledgeable in the domain and who can use an existing expert system framework to extend the capabilities of the evolving system.

The team also requires someone with a detailed understanding of the design and implementation of expert systems—someone who can construct the underlying framework in which to encode domain knowledge. Unfortunately, such people are currently scarce and in demand. Among the ways around this bottleneck are use of off-the-shelf development tools, and training of existing staff in expert system design techniques. Companies presently exist to perform training. Development tools are somewhat more problematic, but they too have started to appear. We will return to this point later in the article.

Finally, the development team requires traditional programming support for integration into pre-existing systems, for graphical interfaces, and so on. In this catchall category we include expertise in related areas (*e.g.*, statistical algorithms and signal processing algorithms) as dictated by the application domain.

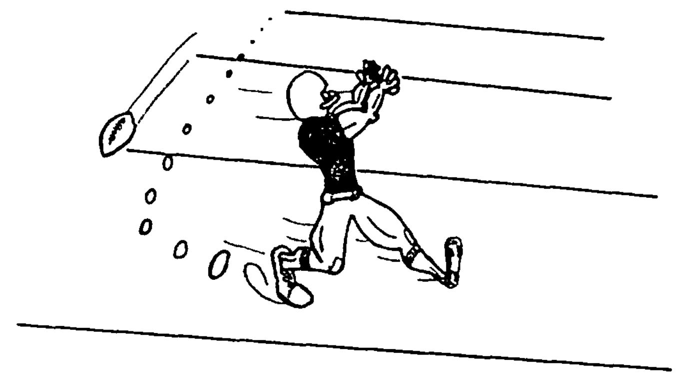
Naturally, some of the skill categories shown may be colocated in the same persons. (This has traditionally been the case for knowledge engineering/expert system tool design.)

In later stages, experiment designers, software engineers, and other domain practitioners (not necessarily experts) are required to test and debug the knowledge and framework in more stringent and wide-ranging tests and to produce the actual commercial product. Once again, it is possible that these tasks will fall to the original team members. We would argue, however, that the downstream engineering of the original code is not the best utilization of people with expert system design skills. These people are too few in number today to be underutilized.

Rapid Prototyping and Successive Refinement

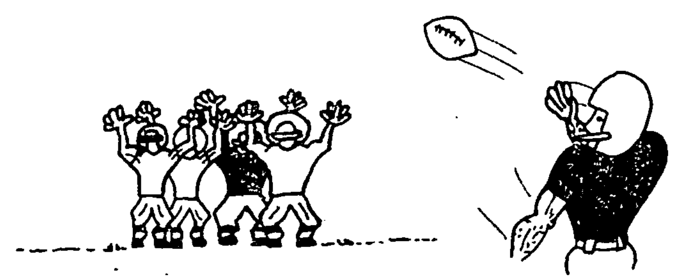
In the beginning of a commercial expert system development project, it is important to demonstrate the feasibility of the system. Rapid prototyping seems to be an appropriate strategy—especially given the usual vagueness of the understanding of what can be accomplished.

The main concern in such an approach is a flexible and powerful development environment. Traditionally, such an environment is not even closely related to the commercial computational environment. This lack leads to the problems noted above. With the advent of inexpensive personal workstations, however, there is real hope that the situation may be changing (as has been our experience with the Dipmeter Advisor system).



Research and Engineering do not have the Same Pattern in Mind

Figure 5.



Research does not appreciate the Constraints under which Engineering operates

Figure 6.

Significant questions still remain. One of the problems of rapid prototyping is that it provides a good start toward system development, but does not offer clear guidance on how to produce a well-engineered commercial product (see, for example Sheil, 1983). Traditionally this fault is viewed as a problem in technology transfer.

The Dipmeter Advisor system has been developed using rapid prototyping techniques and has evolved as a series of prototypes through successive refinement. Our experience with the process suggests technology transfer through not a single release from research to engineering but rather through successive releases, corresponding to successive prototypes. This type of transfer is appropriate for an expert system in which domain knowledge expansion and refinement can be expected to continue for some time, but for which the system framework has demonstrated that it is sufficiently powerful to warrant engineering effort.

Such an approach to technology transfer naturally imposes restrictions. The designers must somehow convey to their engineering organizations a more accurate perception of the expected lifetimes of the prototypes. Furthermore, the designers are forced to pay even more attention to user interfaces than our earlier figures would suggest. If the systems are going to be changing rapidly then they must have especially convenient and easy-to-learn interfaces.

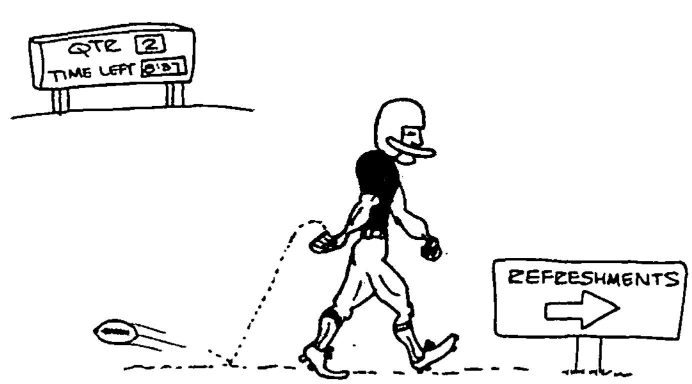
Expert Systems Technology Transfer

Construction of expert systems requires skills that are possessed by a very small number of individuals. Furthermore, the rapid prototyping development methodology makes traditional technology transfer more difficult—the systems are in a constant state of flux. As a result it is fair to say that for the foreseeable future, greater than normal responsibility will lie with the research and advanced engineering organizations to ensure successful transfer.

Based on our experience with the Dipmeter Advisor system, we can suggest some actions to ease the problem. The suggestions refer to a number of phases of expert system development—from problem choice to transfer to engineering.

Technology transfer can be viewed as a (forward!) pass from research to engineering. In order to ensure a successful completion, both passer and receiver must have the same pass pattern in mind. From the point of view of the passer, if no open receivers are open, then a pass is ill-advised. Similarly, the passer must be sensitive to the constraints under which the receiver operates. Throwing the ball in the general area and hoping that a receiver will appear to make the catch is also ill-advised. From the point of view of the receiver, once the ball has been caught, it is his responsibility to move on down the field.

From our football analogy, as represented in Figures 5, 6, and 7, we can take away a number of useful suggestions. First and foremost, for a research organization, constructing demonstrations or prototypes and simply throwing them to engineering isn't enough. The engineering staff need to be aware of the design desiderata, the false starts, the simplifications and approximations made in the interests of expediency, the interactions between components, and so on—a host of insider information. Furthermore, once in the engineering organization, there must be committed and capable receivers to carry the project toward commercial



Engineering is not fully committed to a Successful Transfer.

Figure 7.

deployment.

This need argues for early, on-site involvement by engineering (and perhaps field) personnel in the research laboratory. Such involvement is not atypical for development projects, of course. It is especially important for expert systems technology, however, given its relative immaturity and the lack of trained specialists.

These people can also help to ensure that the completed system is well-integrated into the overall complex of systems in use by the organization as a whole. Furthermore, they are probably in a much better position than the researchers actually to effect this integration.

We have made the point several times that expert system development is an incremental process. Even so, it is worth reemphasizing. Without domain expertise there is no system! Over the course of a commercial expert system development effort, it is necessary to maintain the commitment of at least one domain expert—personal commitment and corporate commitment. The importance of the latter should not be overlooked. Experts make money for their employers and time devoted to a development project may well have a short term negative impact on their normal productivity. Hence, management needs to support expert involvement.

One way to ensure this commitment is to work on problems that the experts actually want solved!

We have found it useful in easing our interactions with both engineering and field personnel to deal in what we might call value-added systems. By this we mean that the ultimate user gets a number of advantages from using our new systems—one of which is symbolic inference. This advantage is evident in the Dipmeter Advisor system. Even if the interpreter never uses the inference machinery, he still derives some benefit from the system—namely, a powerful interactive log interpretation environment. In addition, he is always in control of interactions with the system—he can interactively control the system's inference procedure. This option has the effect of giving him an environment in which he can explore the ramifications of his own hypotheses about the local geology in addition to acquiring access to some of the expertise of other senior interpreters.

We have also found it necessary to construct our systems in such a way that they do not have a negative impact on the standard field computing environment. As pointed out previously, personal workstations have offered real relief in this area. They have, not however been without cost—they have necessitated a relatively large investment in networking software.

For additional thoughts on the problem of moving advanced computer science technology into real world environments see (Newell, 1983). Newell makes a number of salient observations on the basis of his experience with the installation of the ZOG system on the USS CARL VINSON. One of the considerations for which he argues is flexibility. The functionality expected of a system often changes over time. It may therefore be difficult to predict what its eventual use will be. As a result, the developers of real systems are advised to avoid rigidity in their designs.

Some Observations on the Traditional Wisdom

For the remainder of this section we consider a number of maxims of expert system development in the light of our experience in the commercial environment. [See (Barstow, 1981); (Buchanan, 1982); or (Davis, 1982) for good summaries of the traditional wisdom of expert systems development.]

A common maxim of expert system development is that we should throw away the code for the Mark-I version of the system as soon as it demonstrates feasibility and get started on Mark-II. In the commercial environment, there is great reluctance to throw away code. As a result, a likelier scenario involves a series of progressive releases of the system to the expert and possibly to the engineering organization for development and use. The fact is that even though the knowledge engineer knows all too well the limitations of Mark-I, and even has ideas on how to overcome them, Mark-I may still provide some useful service. We do not yet know how to manage this type of progressive and evolutionary technology transfer.[8]

It is well accepted that expert system development is an incremental process. Usually we understand this fact to mean that the performance of the system improves incrementally. There is, however, another kind of change that may occur—namely, our experts are themselves moving targets, partially as a result of the perspective gained through experience in expert system development! This has been apparent during the Dipmeter Advisor project. The existence of tools for testing the ramifications of geological hypotheses led our expert dipmeter interpreter to try a number of approaches to stratigraphic analysis. The program was a test bed for his evolving ideas.

It is traditional wisdom that the task should be very carefully defined before the system is designed. Our experience has been that this process is quite difficult. In consonance with our comments on the rapid prototyping development strategy, it is not clear that task definition can be done in a rigorous fashion. We suggest a contingent definition—one that is clear for a time, but can be easily changed. We should note that the evolving performance of the system itself at least partially fuels changes in the task definition.

It is generally accepted that construction of the Mark-I system should be commenced as soon as one example of the intended behavior is understood. We did not obey this maxim. We now believe that we spent too much time in knowledge acquisition before actually starting to build a

[8]This is a good illustration of a conflict that can arise as a result of somewhat different goals of research and of development in expert systems. The former is concerned with continued exposition and machine implementation of human expert reasoning methods, while the latter is concerned with construction of products that utilize already understood and implemented methods.

system. This activity had the effect of slowing our rate of progress. We could not move forward in formalizing the knowledge that had been gained, because we could not demonstrate in concrete terms our understanding of it.

This problem can be exacerbated by the seductive simplicity of an intuitively appealing formalism like production rules. We want a domain expert to communicate as much as possible about what he is doing in solving a problem—independent of whether or not it appears to him to fit naturally into the formalism. Difficulties can arise when the expert attempts to map his explanations directly into the formalism—perhaps at the expense of accuracy. If insufficient testing is done throughout the process of knowledge acquisition, then a misunderstanding may develop about exactly what the system can and cannot do with rules so generated.

Some of the development team also deemed themselves to have acquired more expertise than was warranted. This is a natural tendency. It was partially due to infrequent interactions with the expert. More responsibility fell on the shoulders of the system developers to organize the domain knowledge than appears prudent. This infrequency also led to a problem of validation—how to be sure that we were on the right track. On a related note, we can testify to the necessity of an adequate set of generic examples with which to test the system as it evolves.

One piece of traditional wisdom might be questioned. It is common to deal with a single expert during the development of an expert system. The perceived danger is that it is difficult enough to capture the perspective of a single expert, let alone those of a number of experts. In the particular context of dipmeter interpretation, however, it might have been useful to involve a number of experts with differing backgrounds from the outset. For example, while the rules for a first approach are most appropriately phrased by a dipmeter interpreter, the necessary geological vocabulary and structure are most appropriately obtained from a geologist. In future systems, we will attempt to synthesize these overlapping points of view.

In a similar vein, we have noted a difficulty that can arise when a single expert is used and when he provides all examples with which to test the system. When working with familiar examples, our expert does indeed appear to apply forward-chained empirical rules—kinds of compiled inferences. Recently, however, we have participated in experiments with a number of interpreters (and examples) from around the world. During these experiments we noted that all the experts exhibited a different mode of reasoning when faced with completely unfamiliar examples. They appeared to reason from underlying geological and geometric models—supplementing the rules. In some sense, this behavior is of course to be expected. However, actual evidence of a change in reasoning style by the single expert that we dealt with for the rule base development was elusive. It was complicated by the fact that he has extremely broad experience. Hence, finding a completely unfamiliar example was quite difficult. We believe that dealing with multiple experts would have provided concrete evidence of this phenomenon much sooner in the life of the project.

With regard to acceptance of the expert systems approach, our experience has been somewhat different from that of the R1 designers in that there was general relatively rapid acceptance of the ideas within our organization. From early in the project, concerns revolved almost totally around performance and utility in the problem domain.

We have seen a substantial increase in the size of the rule base (approximately tripled) and the functionality required of the system before we could consider field evaluation. McDermott has described a similar experience with R1. The size of its rule base tripled during the development phase (McDermott, 1981).

The traditional wisdom notes the importance of early construction of a flexible user interface. For the Dipmeter Advisor system the interface is graphical. It has proved invaluable in testing and user acceptance. Furthermore, as has been noted elsewhere (Buchanan, 1982), expert systems that are actually used by people trying to solve problems in their own domains of interest (as opposed to being used by researchers as vehicles for experimentation with AI techniques) must pay particular attention to human interface issues. For the Dipmeter Advisor system, it was only after we constructed a personal workstation implementation that was flexible, robust, and fast that it became possible to consider seriously testing by the Schlumberger engineering organization.

One final observation worth noting relates to the impact of an expert system on the domain experts. As has been found in other applications of expert systems (Feigenbaum, 1980) the existence of an expert system is helping to identify the real knowledge used in the field—the kind of knowledge that is rarely found in textbooks. A program that captures some of it at least gives a concrete basis for comparing the methods of different experts. As Gaschnig has noted (Gaschnig, 1982) it can also help a group to reach some form of consensus. The Dipmeter Advisor system has stimulated an examination of current dipmeter interpretation methods that promises to improve quality.

System Development Tools

We have discussed the utility of flexible programming environments and personal workstations in the development of expert systems. Powerful tools for creation, modification, and maintenance of knowledge bases and related code are also especially helpful during the design and testing of expert systems. In a sense, they are augmentations to the programming environment that further assist in the rapid prototyping approach. Such tools may also help in reducing the necessity for a tool designer on every expert system development team.

In this section we describe a small set of development tools that we are finding helpful in our current expert system efforts. The tools are STROBE, a structured object programming system, and IMPULSE, a display oriented knowledge base editor. They are described in (Smith, 1983a),(Smith, 1983b), and (Schoen, 1983).

In recent years, interest has been considerable in structured object representation for the design of expert systems. Within this framework, a programmer can encapsulate packets of knowledge and link them together via a variety of relationships to form knowledge bases. Inheritance of properties through generalization hierarchies is standard.

Programming with structured objects offers a number of advantages. From a conceptual point of view, it is helpful to organize computations around programming objects whose internal structure explicitly reflects that of objects in the real world and whose communication with each other is via messages. It is also helpful to organize data structures according to taxonomic hierarchies and to distinguish between general classes of entity and specific individual entities. This style of programming encourages thought about the structure and interrelations between various packets of knowledge in a system.

From a programming point of view, it is helpful to encapsulate procedure definitions and data definitions. This process leads to modular code and helps prevent inappropriate application of procedures. The behavior of an object in stereotyped situations is defined within the object itself in a set of procedures specific to that object as opposed to being defined in a general procedure buried in an amorphous system. Inheritance of procedure definitions also enhances modularity and storage efficiency. It has the added benefit of simplifying the sharing of procedures.

From an expert system point of view, structured objects help to capture what we might call automatic inferences—the kind of inferences that would otherwise be made by explicit rule application (Nilsson, 1980). For when a system built with structured objects discovers that some object belongs to a known class of objects, then it immediately acquires access (through inheritance) to the body of information already known about the class.

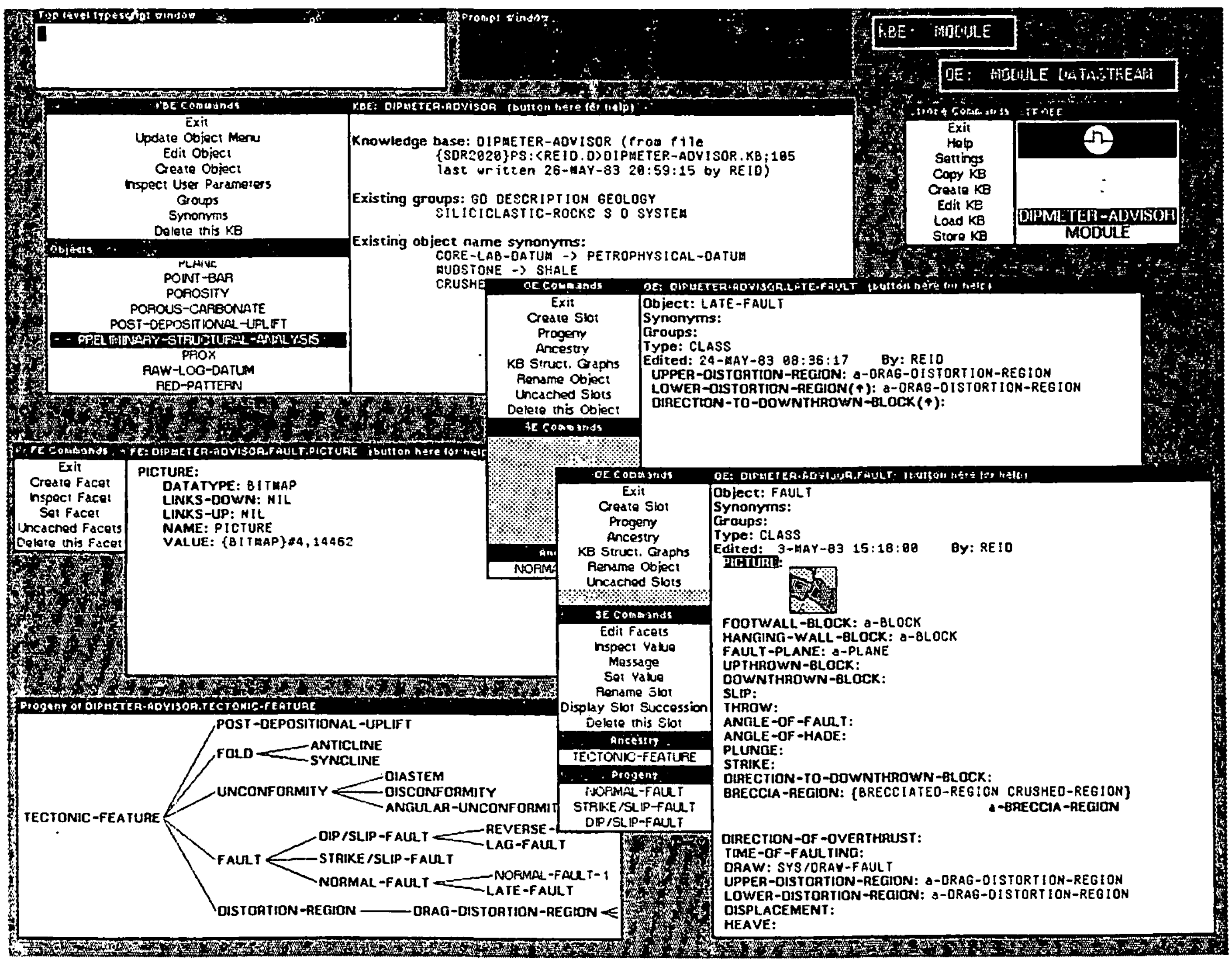


Figure 8. Sample IMPULSE Editor Screen

STROBE is a system that provides object-oriented programming support tools for INTERLISP. A STROBE knowledge base is made up of a number of interrelated objects. The characteristics of an object and its links to other objects are encoded as a number of slots. The slots themselves have structure—facets—that can be used for annotatation. STROBE implements multiple resident knowledge bases, tangled generalization hierarchies, flexible inheritance of properties, procedural attachment, message-passing, and procedure invocation in conjunction with several types of data access and alteration. It offers a primitive foundation with which more complex structured object representation schemes can be constructed.

IMPULSE provides a convenient user interface to STROBE. A user still expresses his knowledge in terms of STROBE constructs, but interaction with the evolving knowledge bases and objects is via pointing and direct visual manipulation. IMPULSE enables concurrent editing in multiple contexts (*e.g.*, having several object editor windows simultaneously active) and graphical displays of inter-object relationships. Figure 8 shows an IMPULSE screen during an editing session in which parts of the tectonic feature knowledge of the Dipmeter Advisor system are being updated.

In addition to their utility to the builder/maintainer of knowledge bases, tools like STROBE/IMPULSE can assist in the transfer of expertise from domain expert through computer scientist intermediary to machine. One of the most useful roles played by the intermediary is to help provide a logical organization for the knowledge of the domain expert. This assistance is typically provided via many interactions. For each interaction, the intermediary gathers some understanding of a portion of the expert's knowledge, encodes it in a program, discusses the encoding and the results of its application with the expert, and refines the encoded knowledge. Discussion and refinement is facilitated when the knowledge is encoded in domain-specific terms and when it is presented in forms familiar to the domain expert. Our experience with IMPULSE is that its ability to simultaneously display different views of a knowledge base and its characteristic immediate feedback have enhanced interactions with our domain experts.

The tools we have discussed are characteristic of the sort that an expert system development team can be expected to use (and perhaps produce). Our focus has been tools for encoding structural relationships between packets of knowledge. They are typically combined with other tools that provide facilities for encoding and invoking heuristic rules. Tools of this sort are described in the literature [*e.g.*, OPS (Forgy, 1981), LOOPS (Bobrow, 1983), (Gorlin, 1981), and EMYCIN (VanMelle, 1981)]

Conclusions

The current Dipmeter Advisor system has provided substantial demonstration of the feasibility of using expert system techniques in commercial well-log interpretation. Additional analysis and evaluation of the system will certainly further define the the strengths and weaknesses of its approach. The experience gained to date has also helped to suggest characteristics of commercial expert system development as well as properties of a development methodology.

We have found that incremental development of expert systems within a rapid prototyping framework is a viable approach. It has also been important to bear in mind from the beginning of a commercial expert system development effort that the system will eventually be used by people who are more sensitive to utility and performance than to the novel techiques that it may embody.

Early engineering and field involvement are especially important in expert system technology transfer, given its relative immaturity and the scarcity of trained specialists. These people are also in a good position to ease the problem of integration of expert system components into more traditional software systems. The notion of value-added systems—that include symbolic inference as part of an overall package—is useful in ensuring field acceptance.

A variety of skills are required for expert system development. These skills domain expertise, expert system tool design, knowledge engineering, and traditional programming support. Not all of these skills are required throughout a development project, as it oscillates between domain knowledge expansion and redesign of the underlying framework. Furthermore, high-level development tools such as structured object representation languages and standard rule interpreters can reduce the need for tool design.

Finally, we have noted that the traditional wisdom of expert system development offers sound advice. Problems to be wary of are related to the seductive nature of a simple formalism and to the extended use of a single expert.

References

Barstow, D.R. and B. G. Buchanan (1981) Maxims for Knowledge Engineering. SDR AI Memo No. 10, Schlumberger-Doll Research.

Bobrow, D.G. and M. J. Stefik (1983) The LOOPS Manual. Unpublished Memorandum, Xerox Palo Alto Research Center.

Buchanan, B.G. (1982) New Research On Expert Systems. In J. E. Hayes, D. Michie, and Y-H Pao (Eds.), *Machine Intelligence 10*. New York: Wiley & Sons, 269-299.

Clancey, W.J. and R. Letsinger (1981) NEOMYCIN: Reconfiguring a Rule-Based Expert System for Application to Teaching. *Proceedings of IJCAI-81*, 829-836.

Davis, R., H. Austin, I. Carlbom, B. Frawley, P. Pruchnik, R. Sneiderman, J. A. Gilreath. (1981) The Dipmeter Advisor: Interpretation of Geologic Signals. *Proceedings of IJCAI-81*, 846-849.

Davis, R. (1982) Expert Systems: Where Are We? And Where Do We Go From Here? AI *Magazine*, Vol. 3, No. 2, 3-22.

Feigenbaum, E. A. (1980) Knowledge Engineering: The Applied Side Of Artificial Intelligence. Tech Rep. STAN-CS-80-812 (HPP-80-21), Dept. of Computer Science, Stanford University.

Forgy, C. L. (1981) OPS5 User's Manual. CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon University.

Gaschnig, J. (1982) Application of the PROSPECTOR System to Geological Exploration Problems. In J. E. Hayes, D. Michie, and Y-H Pao (Eds.), *Machine Intelligence 10*. New York: Wiley & Sons, 301-323.

Gershman, A. (1982) Building A Geological Expert System For Dipmeter Interpretation. *Proceedings of the European Conference on Artificial Intelligence*, 139-140.

Gorlin, D.,F. Hayes-Roth, S. Rosenschein, and H. Sowizral (1981) The Language Reference Manual. N-1657-ARPA, The Rand Corporation, Santa Monica, CA.

McDermott, J. (1981) R1: The Formative Years. AI *Magazine*, Vol. 2, No. 2, 21-29.

Newell, A. (1983) An On-Going Case Study In Technological Innovation. In L. S. Sproull and P. D. Larkey (Eds.), *Advances In Information Processing In Organizations*. Greenwich CT: JAI Press.

Nii, H.P., E. A. Feigenbaum, J. J. Anton, and A. J. Rockmore (1982) Signal-to-Symbol Transformation: HASP/SIAP Case Study. AI *Magazine*, Vol. 3, No. 2, 23-35.

Nilsson, N. J. (1980) *Principles of Artificial Intelligence*. Palo Alto: Tioga Publishing Co.

Reboh, R. (1981) Knowledge Engineering Techniques and Tools in the PROSPECTOR Environment. Technical Note 243, SRI International, Menlo Park, CA.

Schoen, E. and R. G. Smith (1983) IMPULSE, A Display-Oriented Editor for STROBE. *Proceedings of AAAI-83*, 356-358.

Schlumberger (1981) Dipmeter Interpretation: Volume I—Fundamentals. 1981.

Sheil, B. (1983) Power Tools For Programmers. *Datamation*, 131-144.

Shortliffe, E. H. (1976) Computer-Based Medical Consulations: MYCIN. New York: American-Elsevier.

Smith, R. (1983a) Structured Object Programming in STROBE. Research Note SYS-84-08, Schlumberger-Doll Research, (revised March, 1984.)

Smith, R. (1983b) STROBE: Support For Structured Object Knowledge Representation. *Proceedings of IJCAI-83*, 855-858.

VanMelle, W., A. C. Scott, J. S. Bennett, and M. Peairs (1981) The EMYCIN Manual. Tech Rep. STAN-CS-81-885 (HPP-81-16), Dept. of Computer Science, Stanford University.