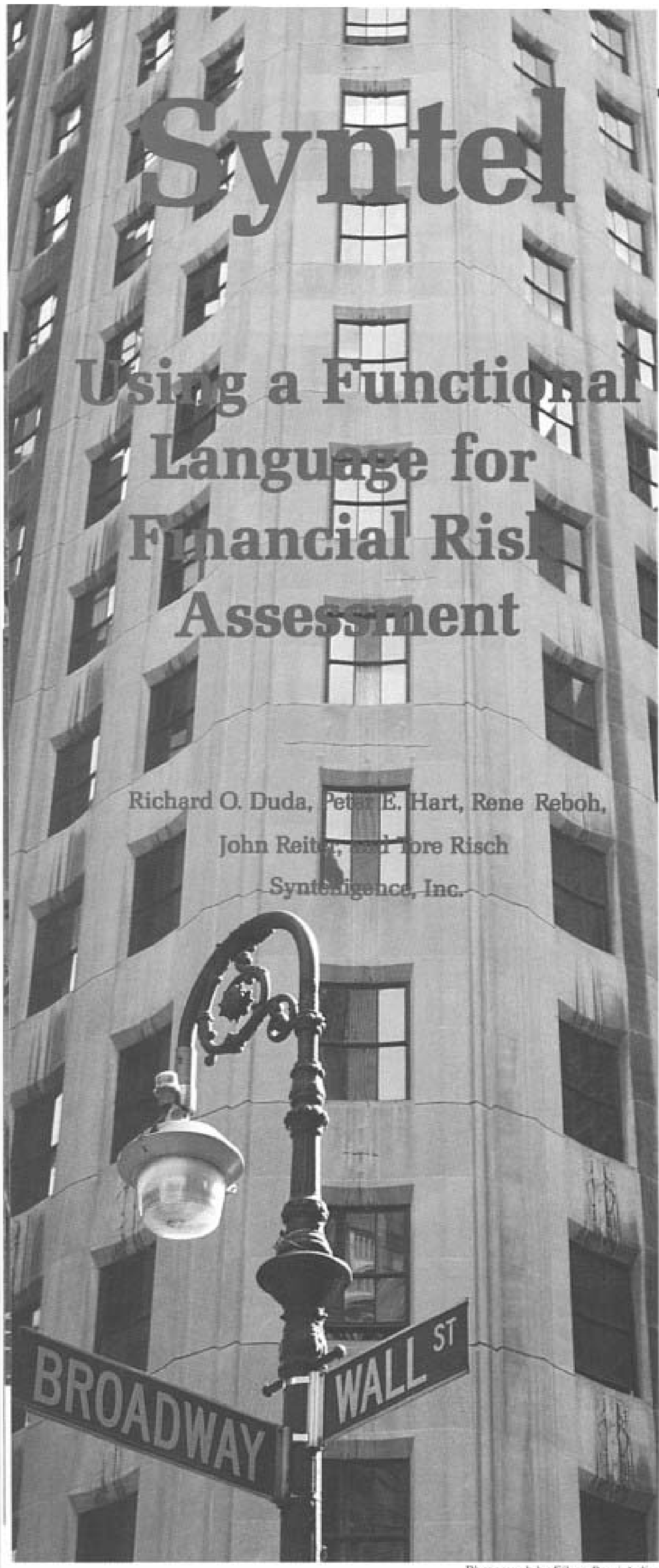
Syntel

Using a Functional Language for Financial Risk Assessment

Richard O. Duda, Peter E. Hart, Rene Reboh,
John Reiter, and Tore Risch
Syntelligence, Inc.



Photograph by Eileen Rassi Ayling

This article describes Syntel, a knowledge representation language used in building large-scale expert systems for financial risk assessment. Syntel is an outgrowth of rule-based systems such as MYCIN[1] and network-based systems such as Prospector.[2] Unlike typical rule- or frame-based expert system shells, however, Syntel is a data-driven, purely functional language providing probabilistic inference plus many kinds of functionality associated with spreadsheet and database systems.

Our use of Syntel exploits both the rapid-prototyping style commonly associated with expert system development, and a more traditional software engineering methodology employing formal specification, design, implementation, and testing phases. The former exploits the AI workstation's flexible development environment to get early end-user understanding and involvement, while the latter provides the discipline and control needed to build commercial-scale knowledge bases.

In either development mode, we implement the knowledge base on Xerox 1100-series workstations using knowledge engineering tools designed specifically for Syntel. We can execute resulting Syntel source code on workstations, or port it to the IBM-based mainframe delivery environment—an environment supporting multiple users efficiently while providing them with access to database and other important information systems.

Syntel incorporates ideas from expert systems research. It also draws heavily on concepts from functional languages, spreadsheet programs, and relational database systems. Specific needs of the application domain called for this combination. Thus, we will begin by describing the application's general nature in sufficient detail to motivate the language features. We will then define the language's main concepts. Finally, we will examine the process of knowledge base development, and summarize the lessons learned from the work.

Financial risk assessment

Financial risk assessment is a major part of many business decision problems. In financial institutions, assessing risk has long been a professional specialty. For example, bank loan officers and insurance underwriters devote most of their time to assessing risks associated with business opportunities. Over time, these professionals develop extensive expertise meriting greater decision-making authority and responsibility than can be given to more junior people. We developed Syntel to capture such expert knowledge naturally, and to disseminate it effectively among less experienced but still professional users.

Surprisingly diverse problems exist in financial risk assessment, and subspecialties have developed addressing each. Even risk assessors specializing in only one area require considerable experience before encountering all the situations that can arise. Despite this diversity, the following common features characterize all financial risk assessment problems:

(1) A mixture of qualitative and quantitative reasoning. Few risk assessments are so obvious that a quick qualitative analysis will suffice, and even the most experienced professional will want to "look at the numbers" before making a decision. However, knowing what numbers to choose—and how to interpret them—requires qualitative analysis.

(2) A natural fit to functional or dataflow languages. One reason that financial people use spreadsheet languages so much more than procedural languages is that spreadsheet languages directly capture meaningful functional relationships between quantitative financial variables. Similarly, we have found that dataflow languages have the same advantages in expressing qualitative relationships.

(3) A combination of limited time and unlimited data. To an experienced professional, each piece of information about a case triggers new questions. Some important questions cannot be answered; others can be answered with further investigation, but each answer's value must be balanced against the cost of getting that answer. Thus, assessment systems must cope with incomplete and sometimes inaccurate information.

(4) Judgmental inputs. While rational risk assessment seeks to make objective decisions by basing risk assessments on factual data, attempting to simulate objectivity in truly subjective situations is foolish. For example, suppose that part of an assessment requires evaluating management's ability to cope with personnel problems. While experts might be able to define measurable factors correlated with such an evaluation, relying on users to make such judgments is often simpler and more satisfactory. Of course, users are often uncomfortable when called upon to make vague, subjective assessments; therefore, assessment systems should provide guidance whenever possible.

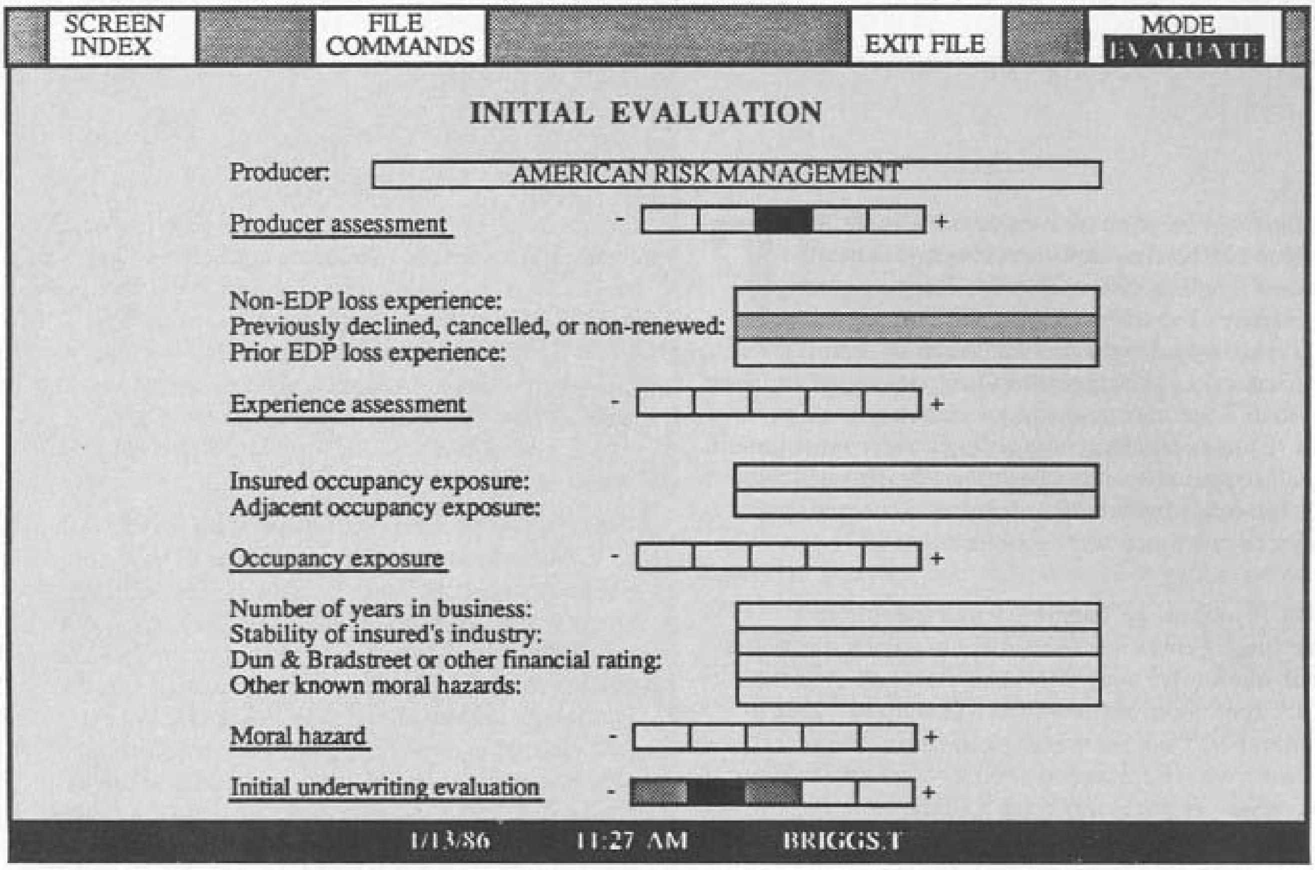
(5) Multi-attribute assessments. Most risk assessment problems involve multiple, interdependent considerations—financial terms, previous financial performance, the management experience of key people involved, possible internal and external changes, possibilities for future business relations, and so forth. Reducing assessments to simple formulas that will withstand ever-changing business conditions and priorities is difficult. Assessment systems must help organize decisions without oversimplifying complex situations.

(6) Tabulated reference information. Professional, governmental, and service organizations have compiled and organized data for many years—data useful for financial risk assessment. Some data, such as hazard ratings for different occupations, are relatively stable; others, such as financial reports for specific companies, continually change. Such reference data are available in numerous databases in either case, and systems must be able to access easily whatever is needed.

These risk assessment characteristics led to many of Syntel's technical features. Other features resulted from the operational requirements of end users and their organizations.

End-user requirements. Daily users want systems to be tools that help organize analysis, provide relevant information, and simplify documentation generation. Most importantly, end users want to be in control. Translated into system terms, this implies data-driven operation. Standard goal-driven consultation "dialogues" (in which systems ask long series of questions that users must patiently answer) may be suitable for infrequent users not familiar with the domain. However, frequent users must be able to control the sequence in which things are (or are not) done.

Corporate requirements. Expert systems change the way work is done, and the costs of changing existing organizations are significant. To the greatest extent possible, therefore, expert systems must fit into standard work-flow patterns. In addition, software must work within existing data processing environments—particularly within existing mainframe-based transaction and database systems. Since corporations may have to change rules quickly, responding to changes in business conditions or corporate policy, they must be able to make and distribute those changes quickly. Finally, all standard software requirements for reliability, accuracy, security, and maintainability apply with equal force to expert systems.



INITIAL EVALUATION

Producer: AMERICAN RISK MANAGEMENT

Producer assessment - +

Non-EDP loss experience:
Previously declined, cancelled, or non-renewed:
Prior EDP loss experience:

Experience assessment - +

Insured occupancy exposure:
Adjacent occupancy exposure:

Occupancy exposure - +

Number of years in business:
Stability of insured's industry:
Dun & Bradstreet or other financial rating:
Other known moral hazards:

Moral hazard - +

Initial underwriting evaluation - +

1/13/86 11:27 AM BRIGGS.T

Figure 1. A risk assessment screen for a commercial insurance example showing a graphical display of uncertainty.

An illustrative example

The following simplified but realistic example shows how Syntel addresses the above requirements. Let's assume the user is an insurance underwriter who must provide property insurance for a medium-sized company, including coverage for its electronic data processing (EDP) facility. Assume further that clerical personnel have entered basic information found on the application form, so that the underwriter can go at once to a screen summarizing all requested coverages. To analyze EDP risk, the underwriter can bring up screens devoted specifically to that analysis.

Initially, our underwriter selects the EDP initial evaluation screen (see Figure 1). Like all Syntel screens, this screen organizes information in familiar business form style—bringing together the following considerations that senior underwriting experts who developed the knowledge base deemed critical to assessing EDP risks:

(1) The producer assessment. How informed and reliable is the primary information source—the agent or broker who submitted the application?

(2) The experience assessment. What are the amounts and frequency of past insurance claims?

(3) The occupancy-exposure assessment. What current activities are taking place in and around the EDP facility?

(4) The moral-hazard assessment. What is the company's financial condition? Is arson a reasonable concern?

Should a few simple questions disclose serious problems in any of these areas, no further time need be wasted investigating other possible concerns.

Figure 1 shows the four key assessments in boxes with a minus sign marked on the left and a plus sign marked on the right. Both input data and output assessments appear in the screen's boxed areas. Although only one input exists at this point (the name of a fictitious "producer"), two assessments are shown—the producer assessment and the initial underwriting evaluation.

Assessments are shown graphically as fuzzy gauges that we call "meters." Meters display an ordinal variable's value as a shaded bar whose position indicates assessment value and whose width indicates uncertainty in the value. By convention, the meter's left side always corresponds to unfavorable assessments and its right side always corresponds to favorable assessments, with the center being neutral. Internally, the system associates probability values with every point on the discrete ordinal scale. Our convention for graphically displaying probabilistic assessments is that solid black bars show 25th to 75th percentiles, and the union of black and gray bars shows 5th to 95th percentiles; users typically interpret black regions as the "most-probable zones" and gray regions as "still-plausible zones." Such graphical displays provide an overall impression of the risk profile quickly, revealing problems at a glance.

Figure 1's producer assessment shows a solid "average," but the initial underwriting evaluation reveals an uncertain conclusion ranging from "far below average" to "average." In this case, the system obtained its producer assessment from database information about the particular producer's profitability and volume. The initial underwriting evaluation shown combines the producer assessment with the other three assessments. Since no input data currently supports these other assessments, the system (1) uses prior probabilities as defaults for the missing inputs, (2) computes probabilities for the five possible bottom-line evaluations, and (3) graphically displays this probability distribution. Thus, while the absence of factual information degrades and blurs the bottom-line evaluation, it does not prevent an assessment from being made.

At this point, users have several options. They can enter input data, override output data, ask for information about any box, or execute any command associated with the command bar at the top of the screen. Both inputting and overriding cause the system to recompute output values on the form. The ability to override enables users to modify conclusions at will, freeing them from having to accept system evaluations.

Let us suppose the initial evaluation sufficiently favorable and the account sufficiently large to warrant more detailed analysis of specific potential concerns such as fire, flood, and security. Figure 2 shows the screen associated with fire risk. Notice that some questions are strictly factual (Is smoking allowed in the EDP room?), whereas others require judgment (How adequate are other fire protective services?). In Figure 2, the menu's bottom half gives typical choices for judgmental questions and the top half gives commands pertinent to input items—the Clarify command, for instance, providing elaborated rephrasing of questions and guidelines for selecting answers. For output items, a corresponding Show Reasoning command provides explanations by searching paths back to supporting inputs. Hiding underneath the menu is the qualitative net-fire-exposure assessment and a quantitative Premises load—the insurance premium portion needed to cover this exposure to risk. These outputs will be important components of the case's final analysis.

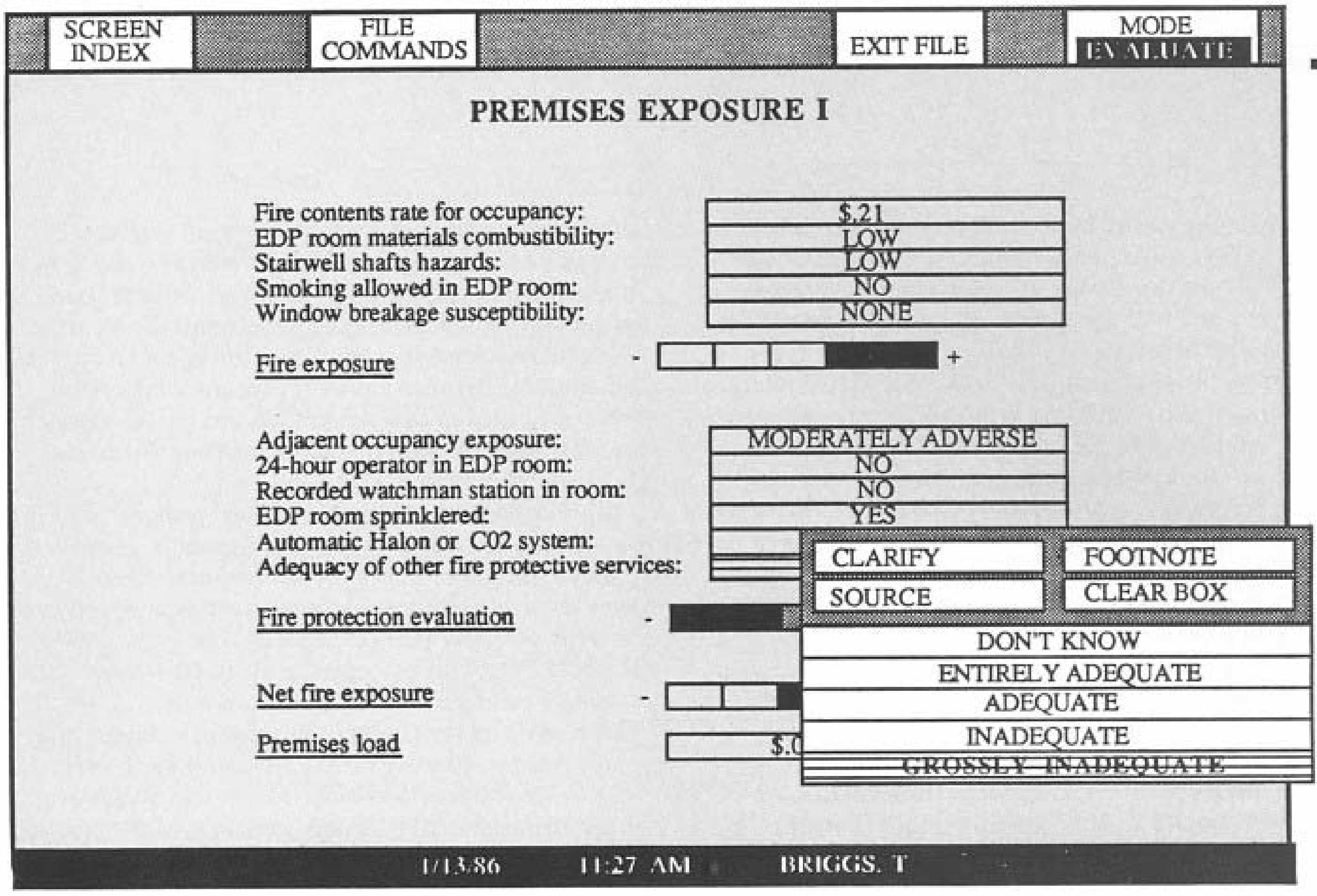
This example illustrates how Syntel responds to financial-risk-analysis system requirements. However, it omits one major feature—the treatment of problems involving multiple instances of the same kind of analysis. Suppose that our fictitious company had several EDP facilities located in several different states. Syntel handles multiple instances by parameterizing the analysis. We can make inputs and outputs depend on one or more instance parameters, such as Site-Number or State. The system makes overall assessments by aggregating over individual instances. Scheduled items in business applications are typically handled by such parameterized analysis.

The Syntel language—an overview

This section describes the Syntel inference engine illustrated in the previous example. We will limit our discussion to basic concepts; Reboh and Risch give a more detailed description.[3]

General characteristics. In general terms, Syntel is a nonprocedural dataflow language.[4,5] Knowledge bases expressed in this language are functional specifications of how input values combine to produce output values. The temporal sequence in which inputs are entered and functions evaluated is of no consequence, since the language is completely free from side effects. In particular, no functions exist for accepting input from or providing output to users. Instead, an independent but tightly coupled (and similarly nonprocedural) *forms language* lets knowledge engineers specify the interface through which users interact with the knowledge base. Both the knowledge base and the forms specification can contain conditional (if/then) statements; however, these conditionals select from alternative data elements, and do not affect control flow.

As our example illustrates, every Syntel variable has an associated probability distribution. Syntel uses a probability mass function for categorized variables and a probability density function for numeric varia-



SCREEN INDEX | FILE COMMANDS | EXIT FILE | MODE EVALUATE

PREMISES EXPOSURE I

Fire contents rate for occupancy:	$.21
EDP room materials combustibility:	LOW
Stairwell shafts hazards:	LOW
Smoking allowed in EDP room:	NO
Window breakage susceptibility:	NONE

Fire exposure - +

Adjacent occupancy exposure:	MODERATELY ADVERSE
24-hour operator in EDP room:	NO
Recorded watchman station in room:	NO
EDP room sprinklered:	YES
Automatic Halon or C02 system:	
Adequacy of other fire protective services:	

Fire protection evaluation -

Net fire exposure -

Premises load $.

CLARIFY | FOOTNOTE | SOURCE | CLEAR BOX

DON'T KNOW
ENTIRELY ADEQUATE
ADEQUATE
INADEQUATE
GROSSLY INADEQUATE

1/13/86 11:27 AM BRIGGS, T

Figure 2. An input menu for an ordinal variable on a screen including various inputs.

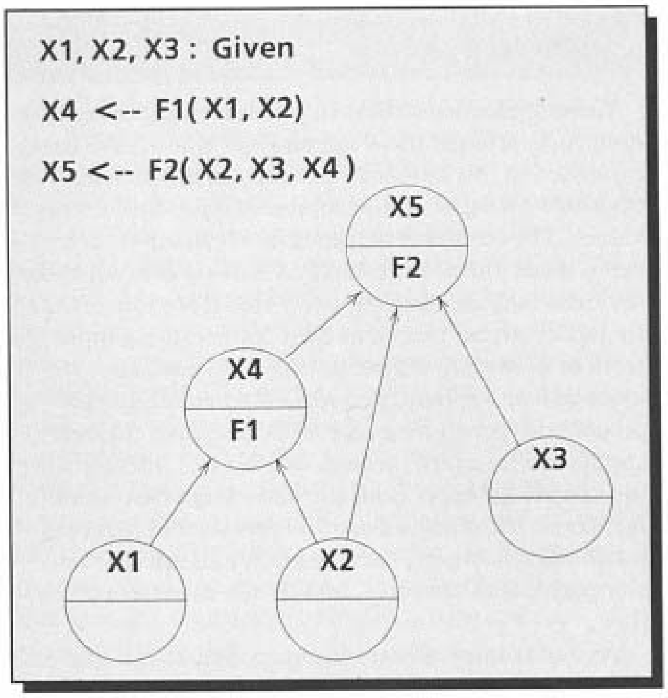


Figure 3. A Syntel function network is a directed, acyclic graph.

bles. These distributions allow inferencing to continue even in the absence of complete input information; as we shall see, they also provide a uniform and well-understood mechanism for expressing uncertainty.

Viewed abstractly, Syntel's most basic objects are variables and functions. Variables are either input variables or functions of other variables. Functions are composed from a predefined library of standard functions. Syntel allows arbitrary functional composition and a restricted form of self referencing, but full recursion (circular dataflow) is not allowed. Thus, given the variables X_1, X_2, and X_3, we can form

$$X_4 \leftarrow F_1(X_1, X_2)$$

and

$$X_5 \leftarrow F_2(X_2, X_3, X_4)$$

but we are not allowed to add, say, $X_1 \leftarrow F_3(X_5)$. If we think of variables as nodes in a directed graph, where arcs point from arguments to functions, then a Syntel equation network must be a directed, acyclic graph (see Figure 3).

In these equations, the "$\leftarrow$"symbol indicates *definition*, not *assignment*—an important point. Proce-

Table 1. A table of values for the variable ConstRisk[Bldg]. Note that each Syntel variable can be viewed as a table in a functional database, the parameters being the keys and the values being the probability distributions. These values, of course, are dynamic and change at runtime. Since the inference engine maintains the relations between variables specified by the knowledge base, the complete set of variables is effectively a data-driven, active database.

Bldg	||	Very-Low	Low	Average	High	Very-High
Main-Office	||	.4	.6	.0	.0	.0
Plant-A	||	.0	.3	.7	.0	.0
Plant-B	||	.0	.2	.4	.3	.1
Warehouse	||			unknown		
Garage	||	.0	.0	.3	.5	.2

dural languages commonly assign the same variable different values in different parts of the program, whereas it would be invalid for Syntel to give the same variable more than one definition. The arrow can also be viewed as indicating dataflow direction, since changes in X_1 or X_2 values can change X_4's value, but a change in X_4 (caused by user override, for example) is not propagated backward to change X_1 or X_2.

Unidirectional propagation also characterizes network-based systems such as Prospector,[6] Agness,[7] and Lucas and Risch's equation-based system.[8] A more general approach would define constraints such as $F(X_1, X_2, X_3) = 0$, without specifying a preferred dataflow direction.[9] However, such generality greatly complicates the value-updating problem.

Syntel variables. At runtime, Syntel behaves like a spreadsheet program, propagating variable values through an equation network. However, Syntel variables are more general than typical spreadsheet variables in at least three ways: (1) their values can be symbolic as well as numeric, (2) they can be indexed by one or several numeric or symbolic parameters to allow for multiple entity instances, and (3) rather than having single values, Syntel variables have associated probability value distributions. By defining successor functions for parameters, knowledge engineers can express parameterized variable values in terms of their preceding instance values; this is the restricted form of self referencing referred to above, and is particularly useful when forming financial projections.

Using a concrete example from the insurance domain, let *Bldg* be a formal parameter identifying a particular building and let *ConstRisk[Bldg]* represent an assessment of the fire risk arising from how a building is constructed. Suppose that a risk assessment value is restricted to the symbolic ordered set { Very-Low, Low, Average, High, Very-High }. Then, at some point, we might represent the variable *ConstRisk[Bldg]* by a table of values (see Table 1).

Thus, we can represent a single parameterized variable's state internally as a set of probability distributions, one distribution for each instance. A knowledge engineer knowing nothing about an instance can either specify a prior distribution of the traditional Bayesian sort or leave the distribution undefined. In practice, while philosophical arguments have yet to be resolved over the proper treatment of ignorance in expert systems, we have found it necessary to provide both alternatives.

Note that each Syntel variable can be viewed as a table in a functional database, the parameters being the keys and the values being probability distributions. These values are dynamic, changing at runtime. Since the inference engine maintains the relations between variables as specified by the knowledge base, the complete set of variables is effectively a data-driven, active, functional database.

As we mentioned earlier, Syntel associates a probability mass function with categorized variables, and a probability density function with numeric variables. Working with arbitrary probability density functions is too expensive for numeric variables; thus, we have used a moment approximation, storing only the mean and the variance (the so-called second-order statistics). While these two moments only crudely describe an arbitrary density function's shape, the variance is a mathematically meaningful measure of uncertainty that usefully exposes the effects of input information that is missing.

Syntel functions. Syntel provides some 55 standard functions for knowledge base construction—functions supplying a set so complete that Syntel provides no escape to the procedural host language. Standard functions fall into four basic families: (1) logical, (2) arithmetical, (3) table and database instantiation, access, and deletion, and (4) voting. Most of these resemble the usual functions provided by any functional programming language. Some are more unusual, however, and a few examples will show how they can be used to represent risk assessment knowledge. For simplicity, deferring the treatment of uncertainty to our next section, let's assume we know all variables exactly.

We can use logical and arithmetical functions alone to implement large parts of a knowledge base. For example, suppose an experienced underwriter knows that fire risk for old frame buildings is very high. This could be represented in rule-like form by the functional composition

ConstRisk[Bldg] ←IF(AND(EQ(*ConstType[Bldg]*, 'Frame), GT(*Age[Bldg]*, 15)), 'Very-High).

Here, the nominal variable *ConstType[Bldg]* identifies the construction type; the numeric variable *Age[Bldg]* gives the building's age in years; AND, EQ and GT are obvious predicates; and IF is the conditional function. In particular, IF(P, V) is a function of two variables whose value is the value of V if the predicate P is true, and is undefined otherwise. Thus, whenever we learn values for *ConstType* and *Age* for the same instance of *Bldg*, the inference engine can use this function to define a value for *ConstRisk [Bldg]*.

We can handle other possibilities by adding more rules and a little more logic to combine them. If the problem has sufficient regularity, a few rules might cover all possibilities. In many cases, however, each possibility seems to need a different rule. In such situations, it is expedient to use the TABLE function

ConstRisk[Bldg] ←TABLE(*ConstType[Bldg], Age[Bldg]; ConstRiskTable)*

where the *ConstRiskTable* might specify Table 2's dependence of risk on construction type and age.

Such tables are convenient when all input value combinations are significant, or when only one output variable is involved. When tables are sparse, or when the same inputs key several output variables, we may prefer to store information in a relational table and use relational database access functions. This mechanism has proven particularly effective for handling specialized knowledge associated with various code systems used in banking and insurance (such as Standard Industrial Classification codes, Insurance Services Organization codes, and National Council on Compensation Insurance codes).

Frequently, the Syntel weighted-voting function expresses assessments naturally through the formula

$$Z \leftarrow v_0 + v_1(X_1) + v_2(X_2) + \ldots + v_n(X_n)$$

where $v_i(.)$ are piecewise linear functions mapping the argument domains to a common numeric range of votes. The special cases where $v_i(.)$ are linear frequently arise in statistical applications as linear regressions, and in financial applications as "weight-and-rate" schemes. The use of additive voting functions presumes some kind of independence of arguments, since votes cast by variable X_i are independent of other variable values. Here, the great virtue is simplicity; one voting function often replaces many rules (or an uninformative table) with something that is logically equivalent but considerably easier both to build and to understand.

Frequently, we must combine the values of different parameterized variable instances. Consider again the variable *ConstRisk[Bldg]*, which gives the construction risk for different buildings. Given such a variable, one might want to sort buildings by risk to find the largest risk, the average risk, and all buildings whose risk satisfies some predicate. These operations typify query functions one would want for a relational database system. Syntel provides 11 functions for such operations. Most of these are conceptually straightforward, but complications arise when parameterized variable values are uncertain. Next, therefore, we turn to the probabilistic treatment of uncertainty.

Probability distribution propagation. The inference engine's basic function is to propagate forward through the equation network the effects of changes to variable values. The state of any variable is either undefined, or defined with some probability distribution on its values. Given an equation $Z = F(X_1, \ldots, X_n)$ and the states of the n arguments, the inference engine must determine whether or not Z is defined and, if it is, must compute the probability distribution for its values. For a parameterized variable, the inference engine must do this for all parameter instances.

In the language of probability theory, separate distributions for the n arguments $P_i(X_i)$ are known as the *marginal* distributions of the joint distribution function $P(X_1, \ldots, X_n)$, and the distribution of a function F resulting from distributions of its arguments is known as the *induced* distribution. In the case of a univariate $F(X)$ function, a general solution exists for computing the induced distribution. Let X be a discrete-valued variable, and let $S_X(f)$ be the set of X values for which $F(X) < f$. Then, the cumulative distribution function for F is given by

$$Pr\{ F(X) < f \} = \underset{X \text{ in } S_X}{\text{Sum}}\ Pr\{ X \},$$

from which we easily obtain the distribution for F (an analogous formula involving integrals holds for continuous-valued arguments). In the multivariate case, we obtain exactly the same formal solution if we substitute the vector $(X_1, \ldots, X_n)$ for the scalar X. Unfortunately, this solution requires that we know the joint, n-dimensional distribution of all arguments $P(X_1, \ldots, X_n)$, not just the n one-dimensional marginal distributions $P_i(X_i)$. Since we rarely (if ever) know these joint distributions, this solution has only theoretical interest.

However, things simplify greatly when variables are statistically independent, so that joint distribution can be written as the product of marginal distributions. For example, consider Table 2 for computing fire risk due to construction. If we want to determine the probability that *ConstRisk* is Average, we note from the Table that two mutually exclusive input situations exist producing that output—new frame construction and old joisted-masonry construction. Thus, we have to compute

$$P(ConstType = \text{Frame}, 0 < Age < 8) + P(ConstType = \text{Joisted-Masonry}, Age > 15),$$

which we cannot determine from the separate distributions for construction type and age. However, if construction type and age are statistically independent, the calculation simplifies to the sum of products

$$P(ConstType = \text{Frame}) * P(0 < Age < 8) + P(ConstType = \text{Joisted-Masonry}) * P(Age > 15),$$

which we can compute from the separate distributions for construction type and building age.

All procedures built into Syntel for computing the induced probability distribution of a function from the probability distributions of its arguments make assumptions about statistical independence. In some cases, these assumptions can be theoretically justified; in others, they are either suspect or demonstrably false. When variables are only weakly correlated, we can often neglect errors generated by assuming independence. However, knowledge engineers basically have only two choices when they know strong correlations exist: (1) reformulate the problem, defining new variables or new network structures that factor the problem into independent components; (2) use Syntel functions that extract probability distributions and build subnetworks that compute desired probability distributions. While the latter solution is available, it tends to produce knowledge bases that are harder to understand. Fortunately, we have rarely encountered situations in practice where the reformulation approach was not possible.

Table 2. A tabular representation of rule-like knowledge about related factors.

	Age		
ConstType	0-8	8-15	Over 15
Frame	Average	High	Very-High
Joisted-Masonry	Low	Low	Average
Fire-Resistive	Very-Low	Very-Low	Low

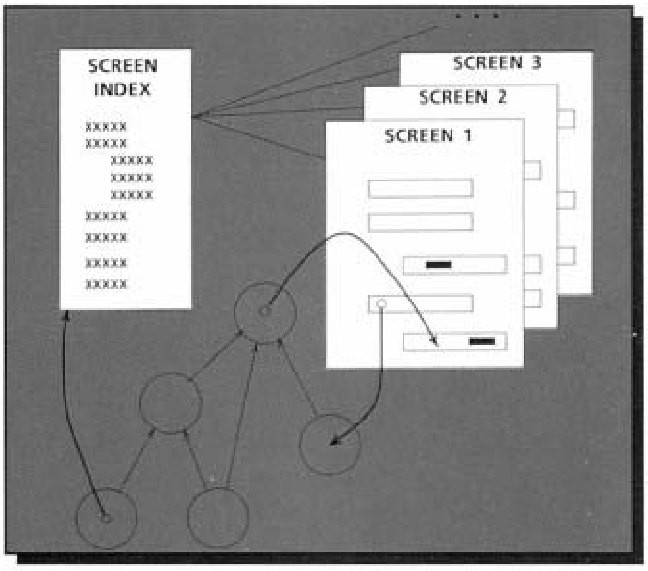


Figure 4. The coupling between screens and the function network.

The Forms system. Of the many factors determining expert system acceptability, the user interface is the most visible. Since all knowledge base interaction takes place through the interface, developers place a high priority on making it as effective and natural as possible. Because we have chosen a business form as the basic display metaphor, we call our interface the Forms system. The Forms system design is complicated by the size of the typical knowledge base, and by the fact that the system is data driven. With goal-driven systems, programs merely have to present users with the one input item of interest to the system. With data-driven systems—particularly one with several thousand variables of potential interest to users—it is more challenging to simplify user access to just what they want.

The Forms system addresses this task by allowing knowledge engineers to define any number of screens, providing viewports into selected parts of the knowledge base (see Figure 4). These screens present an

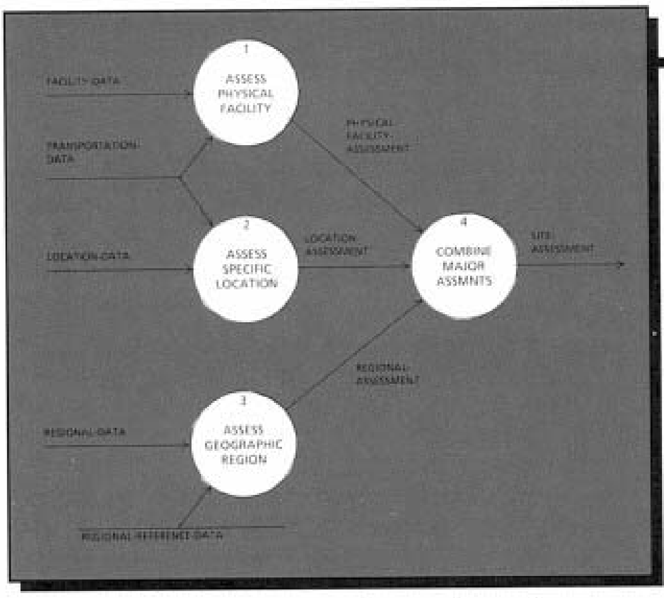


Figure 5. A top-level dataflow diagram for a site assessment knowledge base.

"orthogonal view" of the knowledge base, permitting different ways of grouping variables for different assessment purposes. Using the Forms system's nonprocedural language, knowledge engineers can specify (1) the hierarchical organization of the screens, (2) the layout and format of data elements within any given screen, and (3) the association between these data elements and Syntel variables. The system then takes care of such things as creating a screen index, creating appropriate menus, selecting output formats, justifying layouts, and computing bitmaps.

Frequently, certain questions on a screen or even entire screens are relevant only in particular situations. For example, questions in our insurance example are relevant only when a request exists for EDP coverage. In such cases, knowledge engineers can specify particular regions on a screen or particular screens to be conditionally visible—the conditions being defined by states of variables in the knowledge base. Thus, the initial screen index might be small, providing access only to those screens that define the risk assessment problem. As users enter information, additional viewports into the knowledge base become enabled. Users have complete freedom to move to any enabled screen and to enter or change any displayed item on an enabled screen, but are restricted to what is currently visible.

In addition to providing mechanisms for interacting with the knowledge base, the Forms system gives us a highly effective way to control a well-known weakness of data-driven systems (namely, their tendency toward a combinatorial propagation explosion). At any given time, users view only one screen and can see only a relatively small number of input and output variables. Using compile-time analysis, the system determines for every node a list of screens that node can affect. Whenever users change node values (whether by entering new inputs or overriding old outputs), those changes are propagated through the equation network as long as and only as long as the current screen is affected. Thus, results of changing a node value do not propagate throughout the entire knowledge base, but are blocked at a frontier defined by the current screen. The system defers propagation beyond this frontier until users move to another screen that must be updated. Because Syntel is free from side effects, this method for limiting propagation provides a dramatic increase in performance while maintaining full logical consistency.

Knowledge engineering methodologies

While the application of AI and expert system techniques to software engineering has received considerable attention,[10-13] relatively little has been written on the application of software engineering principles to expert systems[14-16]—probably because the need for disciplined development has been recognized only recently, as large-scale expert systems have begun to enter commercial use.

Knowledge engineering and software engineering. We usually consider knowledge engineering to be the design and implementation of knowledge bases; that is, the representing of knowledge about a particular domain in some knowledge representation language. Unfortunately, this definition emphasizes the static character of knowledge, slighting its dynamic purpose and use. It also supports the myth that knowledge-based systems can be endlessly improved by continually expanding knowledge bases without concern for how various pieces of knowledge will be used or will interact.

We contend that knowledge engineering in Syntel is essentially a form of programming, differing from conventional programming primarily in that it is nonprocedural. Viewed in this light, a knowledge base is a program in a nonprocedural language, an inference engine is an interpreter for that language, and a professional knowledge engineer is a software engineer. This viewpoint emphasizes the purpose and dynamic use of knowledge bases. It also encourages us to learn from past lessons when seeking ways to control cost, manage complexity, and maintain knowledge base quality.

A data dictionary excerpt for Figure 5's dataflow diagram

Shown below is the portion of the data dictionary for the dataflow diagram shown in Figure 5. Dataflows, data elements, and module descriptions are described separately. The input dataflow Facility-data is articulated as several different component dataflows (Space-data, Internal-environment-data, and so forth) whose definitions are found elsewhere in the data dictionary, and which show up in Figure 6. The module Assess Physical Facility is nonprimitive, and is further articulated in Figure 6.

Dataflows

Facility-data	:=	Space-data + Internal-environment-data + Protection-data + Utilities-data + On-site-transportation-data
	:=	...
Location-data		

Data elements

...	
Location-assessment:	Class: Standard five-level-assessment. Notes: Checks included for EPA requirements.

Module descriptions

1 Assess Physical Facility:	Nonprimitive. Combines considerations of space requirements, environmental requirements, protection, utilities and on-site transportation needs.
2 Assess Specific Location:	...

One such lesson is that there is a place for both rapid prototyping and structured methodologies. Rapid prototyping provides an effective way to make progress on unusual new applications. Larger and better understood software projects require top-down design, structured implementation, and similar programming disciplines.

Rephrasing this for expert system development: When neither experts nor end users nor knowledge engineers know clearly what the system will be like, implementing knowledge base fragments helps us obtain something concrete that can be critiqued and quickly revised.

When experts and end users understand what can and cannot be done, and when knowledge engineers understand how to do it, the problem changes from one of feasibility to one of scale. In addition, when completing a knowledge base within time constraints requires teams of knowledge engineers, a uniform and disciplined approach is also required. Under these conditions, knowledge base development shares the same life cycle as other large software systems including such traditional phases as requirements analysis, specification, design, implementation, testing, and maintenance.[14-16]

Fortunately, Syntel suits both research-and-development and software engineering phases of knowledge base development. In research and development, we make maximum use of the programming environment associated with Syntel's implementation in Interlisp-D on Xerox 1100-series Lisp machines. In addition to the powerful facilities inherited from Interlisp, this environment provides interactive editors customized for creating and modifying Syntel objects, graphical means for displaying network structures, and various useful knowledge-base-debugging tools. These advantages of Lisp-based-expert-system development systems are well known, and we will not describe our particular tools in further detail here; instead, the remainder of this section will focus on the less-discussed software engineering phase of knowledge base development.

Knowledge base description and documentation. Documentation is extremely important for all parts of the knowledge base life cycle. In developing our knowledge bases, we have exploited the fact that Syntel knowledge bases directly correspond to the classical Data-Flow-Diagram/Data-Dictionary (DFD/DD) methodology for program specification and documentation.[17] This methodology's central idea is to (1) defer specification of detailed procedural steps stating how output data will be computed from input data, and (2) focus instead on what data transformations are required. Thus, until one reaches the lowest level module descriptions, dataflow diagrams give a nonprocedural program description.

Since Syntel is a dataflow language, the DFD/DD approach provides a natural hierarchical way to describe and document knowledge bases—something essential for knowledge base specification, design, and maintenance. As an illustration, consider the development of a hypothetical knowledge base for site

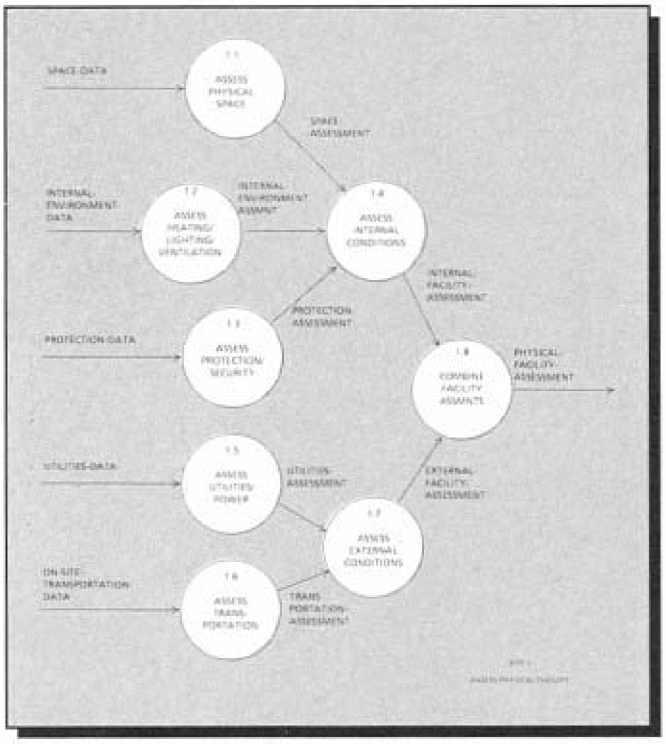

SPACE-DATA
1.1
ASSESS PHYSICAL SPACE
SPACE-ASSESSMENT
INTERNAL-ENVIRONMENT-DATA
1.2
ASSESS HEATING/LIGHTING/VENTILATION
INTERNAL-ENVIRONMENT-ASSMNT
1.4
ASSESS INTERNAL CONDITIONS
INTERNAL-FACILITY-ASSESSMENT
PROTECTION-ASSESSMENT
PROTECTION-DATA
1.3
ASSESS PROTECTION/SECURITY
1.8
COMBINE FACILITY ASSMNTS
PHYSICAL-FACILITY-ASSESSMENT
UTILITIES-DATA
1.5
ASSESS UTILITIES/POWER
UTILITIES-ASSESSMENT
EXTERNAL-FACILITY-ASSESSMENT
1.7
ASSESS EXTERNAL CONDITIONS
ON-SITE-TRANSPORTATION-DATA
1.6
ASSESS TRANS-PORTATION
TRANS-PORTATION-ASSESSMENT
ASSESS PHYSICAL FACILITY

Figure 6. Expansion of Module 1: Assess Physical Facility.

assessment. Let's assume the user is a commercial real estate agent or facilities manager who has to evaluate candidate sites for a small manufacturing plant. Figure 5 gives the top-level view of the entire knowledge base. This DFD shows the four major modules in the knowledge base and their associated dataflows; module and dataflow descriptions are given in the data dictionary (a portion of which is shown in the accompanying box).

At this level, most entering data comes from users, but some comes from database files (namely, the Regional-Reference-Data). We associate no concept of time (or sequencing, or control) with this or any other DFD. In particular, we need not "execute" Modules 1-3 before Module 4 can combine the major assessments. At any given time, some data elements will be known and the rest will be unknown; the inference engine must ensure that whatever output data elements must be computed from current input data elements are in fact computed.

As is usual with dataflow diagrams, modules are either primitive or nonprimitive; nonprimitive modules are further articulated in a standard top-down, recursive manner. The DFD/DD description allows knowledge engineers to descend top down through the knowledge base until they reach primitive modules and primitive data elements. For instance, let's go more deeply into Module 1—Assess Physical Facility. Figure 6 shows that this module is comprised of the following eight modules:

1.1	Assess Physical Space
1.2	Assess Heating/Lighting/Ventilation
1.3	Assess Protection/Security
1.4	Assess Internal Conditions
1.5	Assess Utilities/Power
1.6	Assess Transportation
1.7	Assess External Conditions
1.8	Combine Facility Assessments

While these modules are also nonprimitive, we find a security assessment with the following data dictionary description if we descend further into Module 1.3:

1.3.2 Assess Building Security:
Primitive.
Output(s): BLDG-SECURITY-ASSESSMENT
Weight and combine the following data elements:
- Perimeter surveillance
- Fence alarm
- Door alarms
- Entrance and exit monitoring
- Central control and monitoring

At this point, we have reached a primitive Syntel function—the weighted-voting function. Specifying values for the voting functions is all that remains to complete the description. While we could put all of that information in the data dictionary, it is equally well documented in the Syntel code.

This example shows that a standard DFD/DD methodology provides natural and effective documentation for a knowledge base, providing a description that moves seamlessly from the highest level abstractions to basic Syntel primitives. Of course, it is not the only possible form of knowledge base documentation. For example, we have sometimes found it useful to express the knowledge base in terms of equations. During knowledge base implementation, we often make use of network diagrams like those used for Prospector.[6] In addition, we need separate documentation for screen specifications. However, the DFD/DD approach is particularly well suited to defining knowledge base specifications that can be used during implementation, as well as for testing and maintenance phases of the knowledge base life cycle.

Syntel's current status

Syntel has been in full use since the fall of 1985. Syntelligence installed the first delivered system at

American International Group, a New York insurance company, in the fall of 1986. The Syntel approach has been used to develop several customized versions of five different financial-risk-assessment knowledge bases ranging in size from 800 to over 2000 nodes, with chains more than 100 functions deep and relations containing over 100,000 values. Comparing these size measurements with traditional rule-based systems is difficult since many rules are often required to obtain the behavior of one function, but by any measure these are large-scale expert systems.

Our current delivery environment uses a distributed architecture. The knowledge bases, developed on Xerox 1100-series workstations, use knowledge engineering tools and an inference engine written in Interlisp. The inference engine, written in PL/1, runs on IBM System/370 mainframes under MVS/XA using CICS. The interface software, written in C, runs on 3270 PC/AT workstations. In addition, the mainframe system includes special software to handle database queries and manage business-case data. Cross-development environments are complex; however, they allow powerful tools to speed development while retaining the many advantages of conventional systems for delivery.

We have described Syntel and its application to particular problems in financial risk assessment. Now that this application has passed from prototype to full-scale implementation and daily use, let's review the lessons we've learned. Syntel's nonprocedural, dataflow character has effectively handled most of the risk assessment problems we've encountered. In fact, we believe it applicable to many other problems in assessment or estimation. On occasion, we have excluded kinds of functionality better provided by a procedural language, particularly when the problem involved an arbitrarily complicated search. However, Syntel's freedom from side effects has repeatedly proven highly advantageous—whether for providing override capability, allowing ''what-if'' experiments, retrieving old cases, or merging knowledge bases.

Using ordinal values to represent degree of risk—and probabilities to represent uncertainty in risk assessment—has been effective and readily accepted by our users. In particular, the ability to use prior distributions has been essential since it allows the system to make assessments without experiencing the misleading sense of certainty that comes from simple default values. We can not always assign reasonable prior distributions, however, particularly when numeric quantities are involved; thus, we allow the use of undefined distributions as an alternative expression of ignorance. This blocks conclusions in the face of incomplete information—an undesirable but unavoidable result. In addition, knowledge engineers need to be alert to the assumptions of statistical independence employed by Syntel. When variables are correlated (perhaps because of some common underlying cause), unexpected value pairs should be investigated rather than just combined. Despite these limitations, probability theory provides an understandable and effective way to deal with uncertainty introduced by missing information.

Although an expert system's human interface usually receives less attention than the inference engine, it's critical from the user's viewpoint. Even experts find it easier to think about systems in terms of screen interactions than in terms of dataflows. For business applications, the familiar business form remains the most natural interface. While it doesn't give the impression of a ''thinking computer'' that comes from ''question-and-answer'' dialogue characterizing goal-driven consultation systems, it provides a much more effective mechanism for data-driven systems. Moreover, when the propagation of conclusions is limited to those items visible on the screen, it provides a natural way to control the combinatorial explosion that can otherwise cripple data-driven control.

Finally, our experience verifies the effectiveness of well-known software engineering concepts in the development of large knowledge-based systems. Although expert system technology is new, general principles of managing complexity remain the same. Since Syntel is a dataflow language, it might not seem surprising that the DFD/DD documentation method fits so well. However, DFD suitability can also be viewed as a consequence of separating knowledge bases from inference engines—the standard expert system approach. Knowledge bases tend to be ''data rich'' and are naturally described in terms of dataflows. By contrast, inference engines tend to be ''control rich'' and are not naturally described that way at all. By providing a completely nonprocedural language—not allowing knowledge engineers any kind of procedural escape—Syntel might be viewed as an extreme case. However, we share the belief that DFD/DD methodology can apply to a wide spectrum of knowledge-based expert systems, and expect that it will become a widely adopted form of knowledge base documentation.[15]

As in any computer application, providing good solutions for technical problems is just one requirement for success. In this article, we have focused on considerations specific to expert systems; by and large, we have ignored the work required to implement systems on (and interface them to) conventional data processing systems, and to carry out the many activities needed to put systems into daily use. However, even if these other activities consume most of the total effort, the unique characteristics of expert systems provide the financial industry with many new opportunities for computer use.

Large-scale expert systems are being used daily in financial institutions. The benefits they provide stem from their ability to capture and disseminate expertise. Such easily identified benefits as improved uniformity and consistency, better documentation, and faster institutional reponse justify current developments. We believe that additional benefits, only dimly perceived at present, will eventually emerge.

Formalizing previously unformalized knowledge will provide experts with new insights that will change traditional business practices. As new systems become widely used, the data they capture will provide entirely new views of the "book of business"—an objective basis for strengthening subjective parts of knowledge bases. Recognizing and realizing such benefits will provide a worthy challenge for our creative abilities. ■

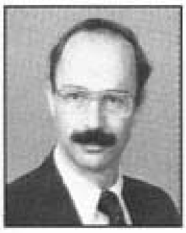
Acknowledgments

Syntel is a trademark of Syntelligence, Inc. Syntel and its knowledge bases were not developed exclusively by the authors, but involved many people over a four-year period. We owe a great debt to our colleagues at Syntelligence whose ideas, encouragement, and hard work made these results possible.

References

1. E.H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York, N.Y., 1976.
2. R.O. Duda and R. Reboh, "AI and Decision Making: The Prospector Experience," in *Artificial Intelligence Applications for Business*, W. Reitman, ed., Ablex, Norwood, N.J., 1984, pp. 111-147.
3. R. Reboh and T. Risch, "Syntel: Knowledge Programming Using Functional Representations," *Proc. AAAI-86*, Philadelphia, Pa., Aug. 1986, pp. 1003-1007.
4. W.B. Ackerman, "Data Flow Languages," *Computer*, Feb. 1982, pp. 15-25.
5. J.A. Sharp, *Data Flow Computing*, Ellis Horwood/John Wiley and Sons, New York, N.Y., 1985.
6. R. Duda, J. Gaschnig, and P. Hart, "Model Design in the Prospector Consultant System for Mineral Exploration," in *Expert Systems in the Micro-electronic Age*, D. Michie, ed., Edinburgh University Press, Edinburgh, Scotland, 1979, pp. 153-167.
7. J. Slagle, M.R. Wick, and M.O. Poliac, "Agness: A Generalized Network-Based Expert System Shell," *Proc. AAAI-86*, Philadelphia, Pa., Aug. 1986, pp. 996-1002.
8. P. Lucas and T. Risch, "Representation of Factual Information by Equations and Their Evaluation," *Proc. Sixth Int'l Conf. Software Engineering*, Tokyo, Japan, Sept. 1982, pp. 367-376.
9. G.L. Steele and G.J. Sussman, "Constraints—A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence*, Aug. 1980, pp. 1-39.
10. Special Issue: Artificial Intelligence and Software Engineering, *IEEE Transactions on Software Engineering*, Nov. 1985.
11. C. Rich and R.C. Waters, eds., *Artificial Intelligence and Software Engineering*, Morgan Kaufmann, Los Altos, Calif., 1987.
12. Special Issue: Building Intelligence into Software Tools, *IEEE Expert*, Winter 1986.
13. D. Partridge, *Artificial Intelligence: Applications in the Future of Software Engineering*, John Wiley and Sons, New York, N.Y., 1986.
14. C.V. Ramamoorthy, S. Shekhar, and V. Garg, "Software Development Support for AI Programs," *Computer*, Jan. 1987, pp. 30-40.
15. R. Keller, *Expert System Technology*, Yourdon Press/Prentice-Hall, Englewood Cliffs, N.J., 1987.
16. N. Martin, *The Software Engineering of Expert Systems*, Addison-Wesley, Reading, Mass., to appear Mar. 1988.
17. T. DeMarco, *Structured Analysis and System Specification*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

Richard O. Duda, a senior scientist at (and a founder of) Syntelligence, is concerned with knowledge representation and inference under conditions of uncertainty. Before joining Syntelligence in 1983, he pursued expert system research at the Schlumberger/Fairchild AI laboratory in Palo Alto, and before that he was a principal contributor to the Prospector system at SRI International. He received his BS and MS in engineering from UCLA in 1958 and 1959, and his PhD in electrical engineering from MIT in 1962. A fellow of the IEEE, he is an editorial board member of *IEEE Expert*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, and *Artificial Intelligence*.



Peter E. Hart is vice president for research and development at (and a founder of) Syntelligence. Before joining Syntelligence, he founded and was the first director of the Schlumberger/Fairchild AI laboratory in Palo Alto, and before that he was director of SRI International's AI center. He received his BS from Rensselaer in 1962, and his PhD from Stanford in 1966. A fellow of the IEEE, he has published numerous papers on artificial intelligence and, with Richard Duda, is coauthor of *Pattern Classification and Scene Analysis* (Wiley-Interscience, 1973).

René Reboh is director of systems development at (and a founder of) Syntelligence, where he is a principal architect and developer of the Syntel system. He came to Syntelligence from SRI International, where he was project director for SRI's expert system group, principal architect and developer of the Prospector and Hydro systems, and creator of the KAS system for knowledge acquisition. He has also done research in AI languages, intelligent database systems, and automatic theorem proving. He received his MS in mathematics and computer science from Uppsala University (Sweden) in 1970, and his PhD in computer science from Linkoping University (Sweden) in 1981.

John E. Reiter is a computer scientist at Syntelligence, which he joined shortly after the company was founded in 1983. There he has developed knowledge bases for commercial property and inland marine insurance, and knowledge engineering tools for the Syntel system. He came to Syntelligence from SRI International, where he helped develop the Prospector and Hydro systems. Before that, he designed and implemented AL/X—a Pascal-based, general-purpose inference engine that was one of the first commercially offered expert system products; while at the University of Edinburgh in 1980, he used AL/X to implement an expert system prototype that diagnosed oil platform shutdowns for British Petroleum. A member of the AAAI and ACM, he received his BS in mathematics and computer science in 1977, and his MS in computer science in 1980, from the University of Illinois at Urbana-Champaign.

Tore Risch is a senior computer scientist at Syntelligence, which he joined shortly after the company was founded in 1983. He is a principal architect and developer of Syntel, having designed and implemented both major components of the system and many of its supporting tools. Before joining Syntelligence, he contributed to both the Prospector and the Hydro projects at SRI International. In 1981, while he was associated with the IBM research center in San Jose, he developed a functionally based knowledge representation language intended for financial and business applications. He did research on application generators, query language compilation, a Prolog database interface, and Lisp program manipulation techniques at the University of Uppsala in Sweden. He received his MS in mathematics and computer science from Uppsala in 1971, and his PhD in computer science from Linkoping University (Sweden) in 1978.

The authors can be reached at Syntelligence, Inc., PO Box 3620, Sunnyvale, CA 94088.

Research & Technology Center

ROBOTICS & COMPUTER SCIENCE

Northrop Corporation, Research and Technology Center, is seeking qualified individuals to be part of a team doing research in conceptual and implementational issues related to robot task planning, geometric reasoning, object oriented programming and data bases, and distributed AI systems. Candidates must have a strong background in AI and a working knowledge in LISP programming.

The Center, located on **Southern California's** beautiful coastline, offers a supportive work environment. **For more information, please call: Dr. H.R. Keshavan at (213) 544-5362 or send resume to: NORTHROP RESEARCH AND TECHNOLOGY CENTER, Employment Office, One Research Park, IEE/0072, Palos Verdes Peninsula, CA 90274. Northrop** provides a comprehensive, competitive benefits package.

U.S. CITIZENSHIP IS REQUIRED FOR SOME POSITIONS. **Northrop is an Equal Opportunity Employer M/F/H/V.**

NORTHROP

Research & Technology Center