The Knowledge-Based Simulation System

Y.V. Ramana Reddy, Mark S. Fox,
and Nizwer Husain, Carnegie-Mellon University
Malcolm McRoberts, McDonnell Douglas Astronautics

To answer what-if questions, KBS uses artificial intelligence techniques to model complex organizations, recognize cause-and-effect relations, and generates scenarios automatically.

Managers of complex organizations are seldom able to answer what-if questions. Systems that simulate such organizations cost too much to develop — and even the ones that are created cannot readily answer simple what-if questions. Instead, a manager must rely on an intermediary, like a systems analyst, for his answers. We created KBS, the Knowledge-Based Simulation system, to address this problem.

The KBS approach is similar to another artificial intelligence simulation system, Ross.[1] Both KBS and Ross are object-oriented modeling systems that contain attribute and behavioral descriptions and provide interactive access and display.

KBS, however, stresses the automatic analysis of simulation results. The model is the kernel of the Intelligent Management System,[2] which must support many functions, including factory monitoring, scheduling, and question answering, as well as simulation. KBS is an interpreter that accesses the model and provides simulation, model checking, and data analysis.

Example domain

Corporate distribution systems serve as example domains to illustrate KBS features. In this example, a manufacturer produces several products made of many components and subassemblies, some of which are produced by the manufacturer at widely distributed locations and some of which are purchased from vendors throughout the world.

These components and subassemblies are transported to several distribution centers where they are stocked. Each distribution center serves customers in its assigned area. The customer may be a retailer or special customer who can deal with the distribution center or the corporate business unit directly.

A customer's product and component requests are processed by the business unit or the distribution center for shipment and merging at the customer site. The distribution centers depend on manufacturing centers and vendors to supply components and products to replenish their stock.

The problem is further complicated by seasonal demands, varying lead times to build or expand manufacturing facilities, need to maintain uniform production levels, contractual agreements with vendors, and effects of weather and labor problems on transportation schedules.

The corporate distribution problem puts tremendous demands on managers at all levels. They face making decisions that have far-reaching consequences to the entire corporation.

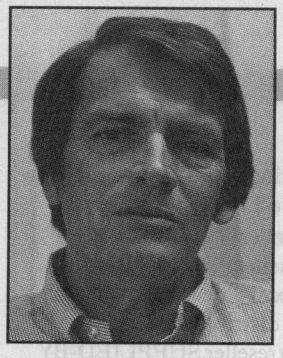
Consider some of the decisions faced by managers at various levels in the corporate distribution system. The primary objective of simulating a corporate distribution system is to answer questions like

- Where should we locate manufacturing plants for various components and what should their capacities be?
- Where should we locate distribution centers and what should their capacities be to meet our forecasted demands in each geographic area?
- Should the products be merged at distribution centers or at customer sites?
- How do transportation modes and schedules affect customer stockouts and satisfaction?
- How do delays in vendor shipments affect key components?
- Do we have enough manufacturing and distribution capacity to meet an anticipated increase in demand for products?
- What are the effects of consolidating manufacturing and distribution facilities?
- How does a proposed order handling procedure affect the corporation?

These questions illustrate the complexity of the distribution domain and suggest the need for tools to aid decision-making at several levels.

For example, low inventories can reduce inventory carrying costs, but frequent

0740-7459/86/0300/0026$01.00 © 1986 IEEE



Robert S. Barton:
language-directed architecture

stockouts can reduce both sales and total profits. Therefore, tools should handle conflicting goals. The KBS approach to simulation deals with such issues. A simplified model of a corporate distribution network is shown in Figure 1.

Model building

A KBS model[3-6] is a collection of Schema Representation Language schemata[7] that represent physical and abstract system entities. The schema is the basic unit that represents objects, processes, ideas, and so forth.

For example, the distribution-center schema (Figure 2) contains slots, some which define its physical limitations (capacity), some which define its current status (inventory), and some which define event behavior (receive-order-event).

Slots can have values, and each may have a set of associated facets or metainformation (printed in italics). The range facet restricts the type of values that may fill the slot. The default facet defines the value of the slot if it is not present.

An important aspect of the Schema Representation Language is that schemata may form networks. Each slot in a schema may act as a relation tying the schema to others. The schema may inherit slots and their values (also called fillers) along these relations.

Model creation in KBS is simply the creation of schemata that represent the model's entities, including specifications for their event behavior and interconnections with other schemata. The window-based schema editor, sedit, offers a fairly general facility to create or alter schemata. One window displays the schema being edited, while the other window accepts commands to manipulate that schema.

Sedit is one of several tools that help model creation and alteration. Once created, models may be perused by visiting each entity or communicating information through pictures.

Model validation. A recurring problem in simulation systems — including KBS — is maintaining model consistency and completeness. We found much time is wasted discovering errors and holes in the model, so we constructed a language and an interpreter to specify model consistency and completeness rules. However, this problem is more comprehensively addressed by the logic programming techniques more recently incorporated in the Schema Representation Language.

A consistency constraint relating the resellers and distribution centers in the cor-

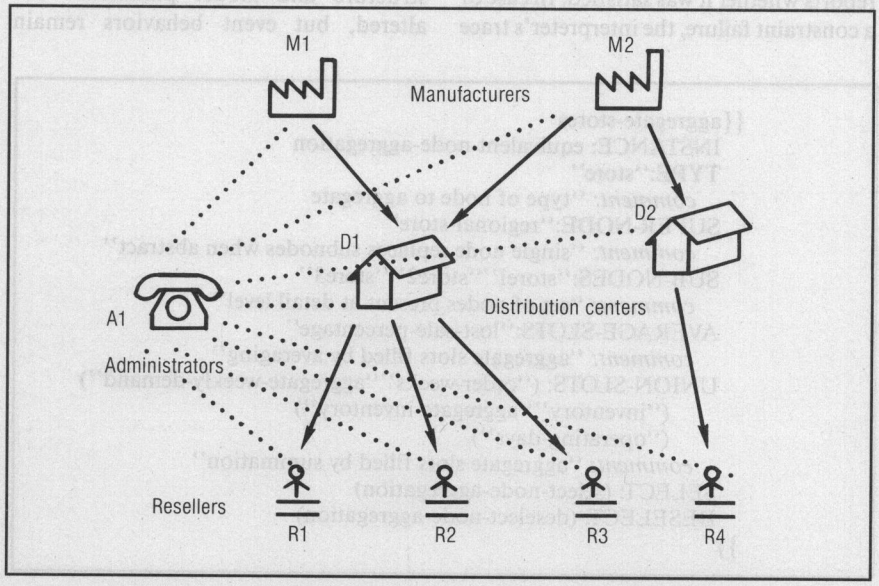


Figure 1. Example of a simplified corporate distribution system model.

```
{{ distribution-center:
   CAPACITY:
   INVENTORY:
   SHIPMENT-TRANSIT-TIME:
   RECEIVE-ORDER-EVENT:"receive-order-rule"
      Range: (TYPE instance event-rule)
   RECEIVE-SHIPMENT-EVENT:"receive-shipment-rule"
      Range: (TYPE instance event-rule)
   ADMINISTRATOR
      Range: (TYPE instance administrator)
   ORDER-TRANSIT-TIME:
      default: 0
   BACKORDERS:
   TOTAL-ORDERS:
   INVENTORY-COST }}
```

Figure 2. Distribution center schema.

porate distribution simulation model may be specified as a Lisp expression.

```
(for-all 'reseller '(VIEWED-AS
instance reseller)
   '(there-exists 'dc '(VIEWED-AS
   instance distribution-center)
      '(and (reseller.SUPPLIED-BY
         = dc)
      (dc.SUPPLIES - reseller))))
```

This constraint may be interpreted as "for all resellers there should exist schemata of the type distribution-center such that the schemata have consistent values for the slots supplied-by and supplies."

KBS evaluates each constraint and reports whether it was satisfied. In case of a constraint failure, the interpreter's trace facility helps determine the source of failure. With the interpreter, we can discover missing schemata and schemata with inconsistent slot values.

Model reduction. Complex simulation models can be simplified or reduced by automatic alterations if the model builder has created a knowledge framework for the task. The simplification makes models run faster, increases model understanding, simplifies analysis, and eliminates unnecessary details.

Model simplification techniques fall into two main categories: static and dynamic. In static techniques, both model structure and model parameters are altered, but event behaviors remain unchanged. Because only the model's static aspects are affected, these are called static techniques. Dynamic techniques alter a model's dynamic processes, redefining some of the event behaviors.

There are two types of static abstraction: equivalent node aggregation and data class aggregation. Equivalent node aggregation combines several nodes into a single node of the same type. This new node must be in some sense the sum of the original nodes. If the new node's parameters are adjusted correctly, the new node will be functionally similar to the old group. The rest of the model is not affected. For example, in Figure 3, individual stores in a region may be grouped to form a regional store.

In data class aggregation, objects are grouped into classes and all references to the members are replaced by references to the class. An example would be to group all items sold into classes. In the computer business, different types of personal computers may be grouped into a broad class called "PC." This greatly reduces the amount of data needed to track inventory levels for each separate type of personal computer.

Dynamic abstraction[8] tries to simplify simulation models by analyzing the stimulus-response event behavior of one or more nodes. It also tries to construct a single node with a stimulus-response behavior that is statistically similar. This technique can combine nodes of the same or different classes.

Statistics on event behavior can be obtained by either constructing stimulus response frames or postprocessing information gleaned from model introspection. However, purely statistical information on events is difficult to use because event parameters cannot readily be abstracted, and the abstracted model may become inconsistent.

```
{{aggregate-stores
  INSTANCE: equivalent-node-aggregation
  TYPE:"store"
    comment: "type of node to aggregate
  SUPER-NODE:"regional-store"
    comment: "single node replaces subnodes when abstract"
  SUB-NODES:"store1""store2""store3"
    comment: "set of nodes present at detail level"
  AVERAGE-SLOTS:"lost-sale-percentage"
    comment: "aggregate slots filled by averaging"
  UNION-SLOTS: ("order-weeks""aggregate-weekly-demand")
      ("inventory""aggregate-inventory")
      ("operating-days")
    comment: "aggregate slots filled by summation"
  SELECT: (select-node-aggregation)
  DESELECT: (deselect-node-aggregation)
}}
```

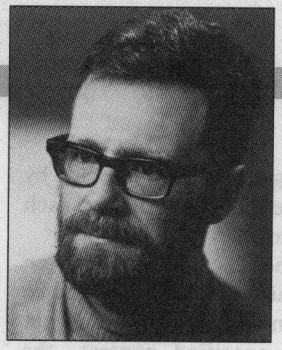
Figure 3. Example of equivalence node aggregation.

```
{{ event-24
  INSTANCE: event-notice
  FOCUS :D1
    comment: Focus of event, the entity
  EVENT :"receive-order-event"
    comment: event-slot in event-schema
  TIME :"21 April 1985 11:00:00"
    comment: Time of execution
  PRE-ACTION: nil
    comment: Action to be taken before event execution
  POST-ACTION: nil
    comment: Action to be taken after event execution
  EVENT-PARAMETER: order10
    comment: Event parameters
  RUN-EVENT: run-event
    comment: method to execute event
  }}
```

Figure 4. An example event notice.

Model dynamics

The KBS kernel interprets the model once it has been defined. KBS uses the discrete event simulation approach, which manages a calendar of events. A calendar is a set of event notices ordered by execution time. An example event-notice schema is shown in Figure 4. In discrete event simulation, the system changes state at discrete points in time. Running the model simply means executing the next



Edsgar W. Dijkstra:
multiprogramming control

event queued in the calendar until there are no more events or until some criteria to stop are encountered.

In the example event notice, the event receive-order-event is to occur on April 21, 1985, at 11 a.m. focused around distribution center *D1*. To execute this event, the simulation clock is advanced by the system to 11 a.m. on April 21, 1985, the value of the slot receive-order-event of the schema *D1* is extracted, and each item found in the list of values is interpreted.

The items in the list of values in the event slot may be representing a Lisp function, a rule, or an instrument designed to collect data, display some text, or produce graphic side effects.

Events. Event behavior may be expressed as rules to be executed when the event occurs. Figure 5 shows an example of an event rule. When interpreted, this rule implements the policy "If there is sufficient inventory to process this order, then schedule a transportation event and reduce inventory by the amount in the order." The order and the object are specified in the rule's parameters. This rule may be deposited in the receive-order-event slot in Figure 2.

Simulation execution. At the end of each simulation run, we may fire a rule base that will suggest changes to the model for the next run. After running many of these goal-directed experiments, we hope to get scenarios that come close to satisfying simulation's goals.

Figure 6 shows the information KBS needs to conduct a series of experiments. INET-expert-run-spec specifies whether introspection, scenario rating, or automatic diagnosis is required. If automatic diagnosis is required, a rule set should be provided. A limit on the number of experiments prevents the system from getting hopelessly lost while trying to find a desirable scenario. At least one experiment should be specified in the expert specification. Users also specify simulated start time, stop time, and active displays.

The KBS-experiment schema in Figure 7 describes individual experiments. The slot model-changes is filled with a set of change specifications that describe the changes that need to be performed on the model before the experiment begins. There are also slots whose values reflect the experiment's status.

Runtime model management. KBS supports a feature rarely found in traditional simulation environments: the ability to conduct a series of experiments without human intervention. Individual experiments don't exist in a vacuum. Instead, they are improvements on past experiments. The improvements are automatically achieved by rule bases detailing the diagnosis and correction heuristics. To work correctly, the rules must allow for reasoning between several model scenarios.

```
{{ receive-order-rule
   INSTANCE: event-rule
   IF: (something-in-inventory)
   THEN: (schedule-transport) (deduct-from-inventory) }}
```

Figure 5. Event-rule example.

```
                    KBS INTERFACE PROCESS
{{ INET-expert-run-spec

    learning-requested: ("no")
    rating-requested: ("yes")
    rules-requested: ("yes")
    experiments-to-be-done: ("e0")
    rule-set: ("default-rule-set")
    learned-database: nil
    iteration-limit: (3)
  }}

umber of experiments before expert system gives up: [5] 3
Is this correct? [yes]
************Starting run 1 in Context g00039
  KBS -- EXPERT SYSTEM COMMAND LEVEL                Wed Oct  9 17:32
```

Figure 6. Filling in the run profile for the expert system.

```
{{ e3
   INSTANCE: "KBS-experiment"
   EXPERIMENTAL-CONTEXT: "kbs-experiment-context-3"
     comment: "context for this experiment, child of base-context"
   MODEL-CHANGES: "e4-change-spec"
     comment: "provides a description of the model for this experiment.
                The base model is altered accordingly"
   .....
   }}
```

Figure 7. Specification of an experiment.

One can manage alternate scenarios by creating a context tree that devotes each context to a scenario.

Contexts in the Schema Representation Language are hierarchically arranged so that if a schema being accessed is not found in the specified context, the context's ancestors are searched until the schema is found or the search fails. A schema may only be created within some context, and the name of the schema must be unique within that context. Schemata with the same name may, however, exist within different contexts and contain conflicting information.

Model introspection. Introspection[9] is acquiring knowledge, or learning, about the dynamics of model execution automatically by tracing. By postprocessing the learned knowledge, we can detect causal chains of events and attributes. Causal chains are formed by events causing other events, events affecting attributes by increasing or decreasing their values, and events accessing attribute values.

The system can detect how an event affects an attribute — but not vice-versa. We therefore assume that if an event accesses the value of an attribute it should be included in the attribute event chain.

Introspected information collected during model execution is stored as metainformation on event and attribute slots. Figure 8 shows an example of entity *D1* after introspection.

Data collection. The task of data collection is concerned with recording the changes in the value of a parameter. This can be done by constantly monitoring the parameter and recording every change or by sampling the parameter. The former yields greater accuracy but incurs greater computational overhead while the latter approach is less accurate but satisfactory in many cases.

In KBS, monitoring is accomplished by attaching demons to slots while sampling is done either by scheduling data collection events or as a separate action during the execution of regular events. Since data collection in KBS is analogous to using measuring instruments, we have introduced the notion of an instrument.

Instrumentation is selective so that only the data relevant to a particular goal may be gathered and analyzed. Although the selective instrumentation facility goes a long way toward simplifying data collection and analysis, it still requires the simulation user to manually instrument each schema and subject the collected data to an appropriate analysis.

```
{{ D1
  INSTANCE:"distribution-center"
  BELONGS-TO:"inet"
  RECEIVE-ORDER-EVENT:
    {{ INSTANCE:"KBS-event"
      CAUSES: (R1 (receive-shipment-event)) ... (A1 (receive-order-event 33))
        comment: "number of times events that are caused by this event"
      CAUSED-BY: (R1 (send-order-event 12)) ... (R3 (send-order-event 12))
        comment: "number of times events that caused this event"
      AFFECTS-ATR: (KBS-generated (new 3))
          (D1 (inventory 3) ... (total-orders 36))
        comment: "describes how many times this event has affected attributes"
      ACCESS-ATR: (KBS-generated (new 78))
          (D1 (inventory 42) ... (order-transit-time 33))
        comment: "describes how many times this event has accessed attributes"
    }}
  INVENTORY:
    {{ INSTANCE:"KBS-attribute"
      AFFECTED-BY: (D1 (receive-order-event 3) (receive-shipment-event 3))
          (A1 (process-order-event 14))
        comment: "described number of times events affected this attribute"
      ACCESSED-BY: (D1 (receive-order-event 42) (receive-shipment-event 6))
          (A1 (process-order-event 111))
        comment: "describes the number of times events have accessed this
    }}    attribute"
  .....
  }}
```

Figure 8. An example introspected schema.

Automatic analysis

One important task for a simulation analyst is to fine-tune the input parameters of a model to bring the values of output variables within a desired range. Rating a scenario[10] measures the goodness or badness of simulation results. For example, if the goal is to eliminate stockouts and if in a given simulation the average stockouts is 10 percent of total orders, we would like a better answer than "no" when asked, "Have we reached the goal?"

To rate scenarios more smoothly, we chose a continuous scale of rating from −1 to +1 in which −1 means the results are far from the goal and +1 means the goal has been completely satisfied.

Since goals are often complex and may consist of conflicting subgoals, we approach the specification of simulation goals as a composite set of constraints[11] on the performance of various entities of the system modeled.

The steps in the construction and evaluation of goals are

- Representing each organizational goal as a set of constraints.
- Selecting and attaching instruments to gather data.
- Specifying procedures for computing performance measures from raw data.
- Executing the simulation for the given scenario.
- Evaluating each constraint by computing a coefficient of constraint satisfaction, which may be positive to indicate the goal has been met or negative to indicate the goal has not been met.
- Evaluating the scenario by computing a coefficient of goal satisfaction as a weighted average of constraint satisfaction coefficients.

Consider a composite organizational goal to increase customer satisfaction while keeping the distribution overheads low. This goal may be broken down into two subgoals: (1) Customer stockouts should not exceed five percent of orders and (2) distribution cost per unit sold should not

Richard W. Hamming: error-correcting code

exceed 10 percent of the manufacturing cost (see Figure 9).

The actual goal evaluation is an event scheduled to occur at some future date according to evaluation-schedule, at which time it is evaluated.

A goal evaluation function will retrieve the contributing-constraints of the organizational goal. Each constraint is evaluated to compute a rating. This rating, weighted by the importance of the constraint, contributes to the overall rating of the organizational goal. If the goal is evaluated more than once, the direction of the successive ratings can help one decide how well or badly the organization is doing.

The INET-goal discussed in the example on rating scenarios was implemented on the example corporate distribution model. After a few days of simulated time, the goal was evaluated and reported (Figure 10). This result indicates that customer satisfaction was bad because it was rated negatively (−0.9), but the distribution costs were economical and were rated at +0.844, which is good. However, as the calculation below shows, the overall goal rating is still negative and therefore unsatisfactory.

Goal Rating = wt. avg. of individual constraint ratings =
(0.3 * 0.844 + 0.7 * −0.900) / (0.3 + 0.7)
= −0.3768

The instruments, goals, and constraints are constructed manually but data collection and reporting are done automatically.

When goals are more complex, they may be viewed graphically with a Kiviat chart like the one in Figure 11. In Kiviat graphs, all performance parameters that are "good" when they assume large values are plotted above the *x*-axis while the performance parameters that are "bad" when they are large are plotted below the *x*-axis. The shape derived by connecting these parameters quickly shows whether the current scenario is good or bad (bad scenarios have large areas below the *x*-axis).

Rule-based diagnosis. Another way to evaluate simulation results is through rule-based analysis. The rules capture the knowledge of expert simulation analysts. The Schema Representation Language provides an integrated rule, logic, and object programming environment that uses a single schema representation. Expert knowledge is codified as rules or logic programs to identify situations such as bottlenecks and lack of inventory. An example rule is

IF a Reseller's average inventory is high
AND Stockouts are high
AND shipments are received infrequently

```
(a)
{{INET-goal
  INSTANCE: "BBS-goal"
  CONTRIBUTING-CONSTRAINTS:"satisfy-customer""economize distribution"
  EVALUATION-SCHEDULE: "daily-at-midnight" }}

(b)
{{ satisfy-customer
  INSTANCE: "goal-constraint"
  CONSTRAINED-BY: "satisfy-customer-spec"
  IMPORTANCE: 0.70 }}

(c)
{{ economize-distribution
  INSTANCE: "goal-constraint"
  CONSTRAINED-BY: "economize-distribution-spec"
  IMPORTANCE: 0.30 }}
```

Figure 9. (a) An example of the corporate distribution system's goal (b) Retailer satisfaction goal constraint. (c) Distribution cost reduction goal constraint.

```
                    KBS ANALYSIS
                GOAL EVALUATION REPORT

GOAL :  INET-goal          CLOCK: Wed Sep 25 00:00:00 1985

Purpose:  to reduce stockouts and economize distribution

      constraint              importance      rating
      ----------              ----------      ------

Inventory Cost                    0.3           0.8
Stockouts                         0.7          -0.9

   Overall Goal Rating : -0.37

MODEL : INET                                Wed Oct  9 17:34
```

Figure 10. Example of goal evaluation.

Ideal simulation environment

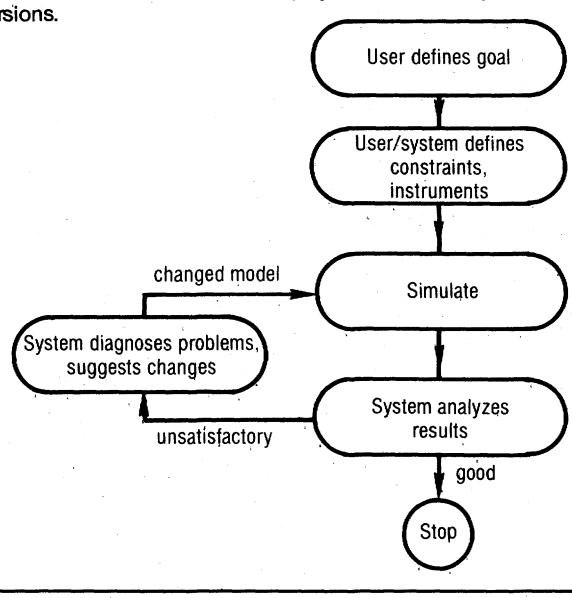
Before creating KBS, we analyzed what was needed to create an decision-support environment. We identified a need for more sophisticated tools to analyze organizations and for tools that managers could use themselves.

We tried to understand functionality types that managers need to construct, execute, and analyze simulations. We also tried to understand how functionality affects the underlying modeling theory, because the model dictates the usefulness of any simulation. We restricted our analysis to discrete event simulation systems and examined many simulation modeling theories and methodologies.

Our research shows that an ideal simulation environment should include facilities for

- declaratively representing models that reduce or eliminate the programming effort;
- different programming paradigms such as object-oriented, logic, data-oriented, and rule-based programming;
- behavioral representation of system entities through an object-oriented (frame-based) knowledge representation that lets entities be altered without altering the simulation model interpreter;
- expression of events as rules to make models more readable;
- alternate command interfaces such as text commands, natural language, and graphics;
- selective instrumentation of models so that only data of current interest is collected;
- automatic model abstraction where the model is represented at multiple levels of abstraction so the user can specify the simulation level and the system in turn automatically configures the model;
- model validation to save time otherwise spent verifying model consistency and completeness;
- interactive access to model building and simulation through a command interface using windows and graphics;
- goal-directed instrumentation where a user need only select a goal so the system can automatically select the appropriate instruments, attach them to the relevant schemata, analyze the data, and report its conclusions;
- model understanding to disclose model behavior by stepping, tracing, and displaying communications between and within events;
- model verification to check whether modeling intentions are carried out faithfully;
- interrupts and checkpoints that can explore the effects of alternate decisions while preserving the original simulation's states;
- automatic scenario rating to assess the desirability of model scenarios;
- rule-based diagnosis that mimics reasoning to both detect problems and recommend solutions to them;
- statistical analysis through a comprehensive library of functions including regression analysis and random number generators;
- causal path analysis to see how much variance each cause contributes to each effect modeled;
- color graphics to depict models statically and dynamically;
- business graphics to display data as plots, bar graphs, histograms, pie charts, gantt charts, and kiviat charts;
- selective report generation to schematically design reports about the simulation's goals; and
- simulations integrated with expert systems to examine the performance of a scenario and suggest model modifications.

While KBS, in its present state, incorporates many of these features, several features such as automatic model abstraction are not yet available. They will be incorporated in future versions.



THEN
indicate possibility of a transportation bottleneck

The example is not an actual rule. Currently KBS's diagnosis and correction rules are written in OPS5. The working memory on which these rules operate is created from Schema Representation Language schemata.

Causal path analysis. The output from simulations is often analyzed by such traditional methods as regression and correlation between the variables in the model. However, we must first determine the underlying causal structures before we can undertake the analysis.

The objective of path analysis[12] is to detect causal relations embedded in a simulation model and exploit that knowledge to generate scenarios that realize organizational goals. Causal relationships between variables can be detected by postprocessing knowledge learned in introspecting simulations. Path analysis research handles both the qualitative and quantitative aspects of rule-based analysis of simulations. A simple rule-based analysis without path analysis is nothing more than automating the ad hoc approach to model simulation.

Typical steps involved in the refinement of rules using path analysis are

- A rule is proposed by the domain expert.
- The causal assumptions the rule is based on are validated against the running model. Validation implies detection of causal chains in KBS models. The rule may need to be modified if the initial causal assumptions were either erroneous or insufficient.
- Alternate causal structures are then proposed and automatically analyzed. The most appropriate causal structure is chosen and the results are summarized to yield an equation reflecting the sensitivity of the output parameter with respect to the controllable input parameters. Thus, path analysis finds the degree to which each particular cause in the modeled system determines the variation of a given effect.
- The sensitivity information is used to quantitatively refine the rules.

A detailed example. A typical example in the corporate distribution domain illustrates the steps taken to arrive at a refined rule after starting from a more

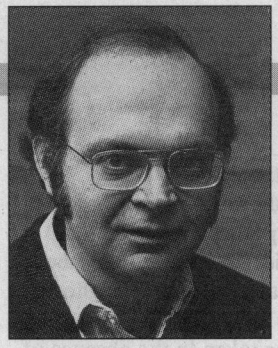


Photo courtesy *Annals of the History of Computing*

Donald E. Knuth: science of computer algorithms

general rule proposed by the domain expert. The rule here is a diagnosis and correction rule.

IF The Goal is to minimize stockouts and Average-Inventory is Low and Stockouts are High and Production Rates can be monitored
THEN
Increase the Production rate

To be specific, the rule must refer to actual instances of objects in the model. We are using the I-Net model of the corporate distribution system domain in Figure 1.

By concentrating on manufacturer M_1 and distribution center D_1, the rule becomes a bit more specific expressed as

IF The Goal is to minimize D1:stockouts and D1:Avg-Inventory is Low and D1:Stockouts are High and M1:production-rate can be altered
THEN
Increase M1:production-rate

Drawing on simple domain knowledge, D1:stockouts is an output parameter and M1:production-rate is an input parameter. A few causal structures, as shown in Figure 12, are proposed that attempt to include the variables in the rule causally connected to each other along different paths.

First, we used KBS's introspection to verify whether D1:avg-inventory, D1:stockouts, and M1:production-rate are indeed causally connected. For example, in hypothesis h1, we must verify whether M1:production rate is connected via an events chain to D1:stockouts and D1:avg-inventory. We must also verify whether D1:avg-inventory is connected to D1:stockouts. Figure 13 shows the causal chain connecting M1:production-rate to D1:stockouts. It is not the only possible path between the two variables.

M1:manufacturing-event accesses attribute M1:production-rate. M1:manufacturing-event causes a send-shipment event. The D1:receive-shipment event receives the shipment, causing an increased inventory at D1, which affects the attribute D1:inventory. The receive-order event accesses attribute D1:inventory to check material in stock. If the inventory is insufficient for that order, D1:inventory records a stockout, which affects the attribute D1:stockouts. Therefore, we assume that M1:production-rate is causally

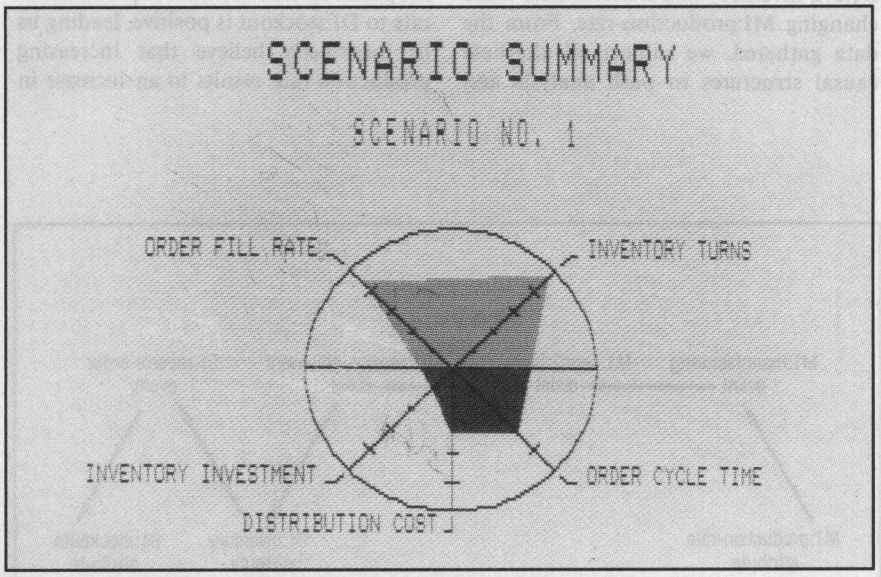


Figure 11. Complex goals viewed graphically.

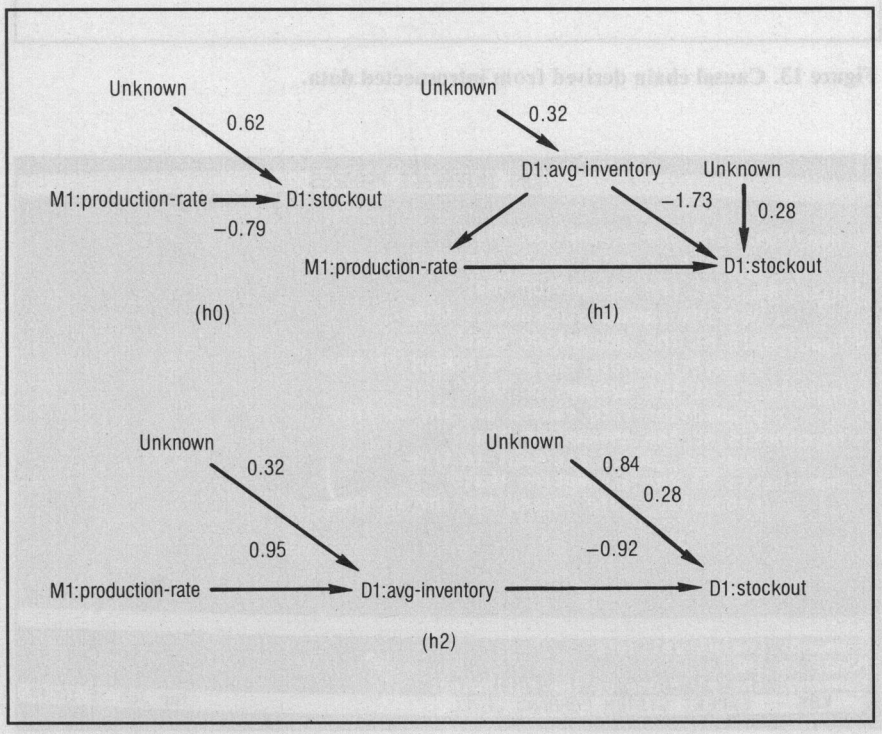


Figure 12. Causal hypotheses in inet-pc model.

connected to D1:stockouts.

As Figure 14 shows, the causal assumptions made in hypotheses *h0*, *h1*, and *h2* in Figure 12 are valid. We included hypothesis *h0* to show the naive approach — ignoring the intermediate variable D1:avg-inventory — that would be taken in the absence of path analysis.

A series of experiments measures D1:avg-inventory and D1:stockouts while changing M1:production-rate. From the data gathered, we subject all alternate causal structures to path analysis and compute the coefficients in the path diagrams.

Figure 15 shows the equations derived from various hypotheses. The residual (unknown) influence on D1:stockouts (+0.62) in *h0* is higher than in *h1* and *h2* (+0.28), suggesting that hypothesis *h0* should be dropped from further consideration. We then reject *h1* because the path coefficient from M1:production-rate to D1:stockout is positive, leading us to incorrectly believe that increasing production rate results in an increase in stockouts. The most appropriate hypothesis is *h2*, and from the unstandardized path equations we derive the sensitivity information necessary to improve the rule.

Summarized path analysis:

D1:stockouts = −0.086
M1:production-rate + 39.96

Sensitivity:

δ(D1:stockouts)/δ(M1:production-rate) = −0.086

δ(M1:production-rate)/δ(D1:stockouts) = −11.6

Getting back to the rule, a possible refinement may be

IF The Goal is to keep D1:stockouts below 40% AND
D1:Avg-Inventory < 30 AND
D1:Stockouts are in the range 40%-50% AND
M1:production-rate can be altered
THEN
M1:production-rate =
M1:production-rate + 11.6
(D1:stockouts − 40)

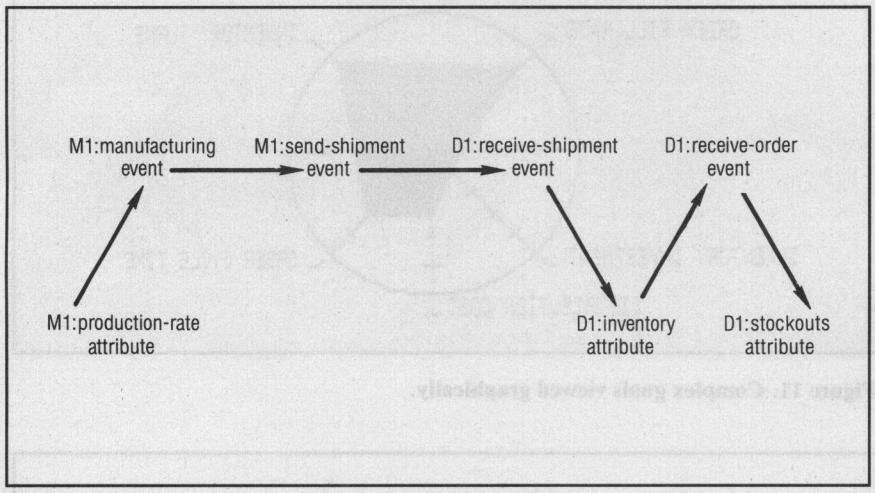


Figure 13. Causal chain derived from introspected data.

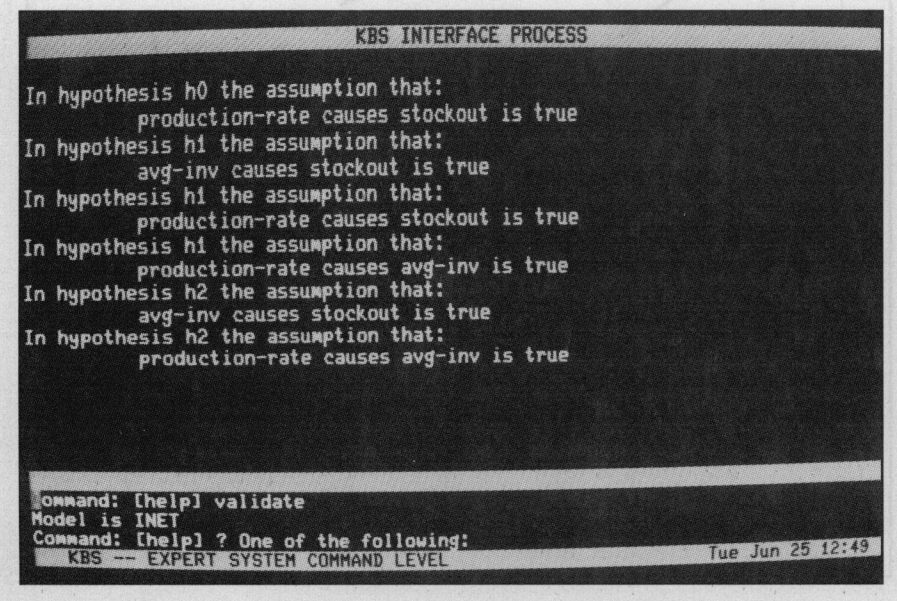


Figure 14. Validating hypotheses.

KBS architecture

Conventional simulation systems are essentially descriptive tools because they only describe the performance of a given scenario. They do not provide any clues as to what changes should be proposed so that the simulation goals may be realized. Figure 16 shows the architecture of a KBS-based expert system.

In this architecture, the user interface processor receives a request from the user. If it is a simple request for information, the model database is accessed to retrieve the appropriate information. If it is a request for a prescription (that is, a goal-oriented request), the request is analyzed by a rule-based goal analyzer.

This analysis may invoke an operations research tool, a specification to conduct a series of experiments, or a specification to execute the simulation model in the learn mode to detect causal relationships.

When causal relationships are detected, they may enhance the domain rule base used in diagnosis. Experimental results are then analyzed. If they are satisfactory with respect to the goal (the scenario achieved a high rating), a recommendation is made based on the best scenario. If the results are not satisfactory, the diagnosis and correction rules are fired. They may generate

Maurice W. Wilkes: microprogramming

change specifications for the next experiment.

KBS can perform several types of analyses to satisfy a specified goal.

Static analysis. Used when no time-dependent information plays a role in the analysis.

Performance analysis. Used when one of the following is requested:

- Scenario rating.
- Diagnosis. This detects the causes of unsatisfactory model behavior.
- Scenario generation. This is used when several scenarios have to be tried before a prescription can be provided.
- Trade-off. This is used when necessary to compare several scenarios.

Learning. If the objective of executing the simulation model is to detect embedded causal relationships, the model is executed in the learn mode and subsequently processed by the causal analysis module.

Mathematical analysis. If the goal analyzer determines that the current request can be satisfied by a mathematical analysis rather than by simulation, it will invoke an appropriate analysis tool.

Figure 17 shows a log of a simulation model execution analyzed automatically. Its automatic analysis led to construction of an improved scenario, kbs-experiment--2. The run specifications that produced this analysis are shown in Figure 6.

Analyses that may be performed by a system such as the one shown in Figure 16 include

Predictive analyses. These address problems such as "What is the effect of adding a new distribution center in Chicago?" The solution is to change model parameters and structure, select performance parameters, execute the model, and report conclusions.

Diagnostic analyses. These address problems such as "What is the cause for delays in transportation to San Diego?" The solution is to use causal path analysis and response analysis.

Trade-off analyses. These address problems such as "To reduce distribution delays in the central region, what is the trade-off between expanding the capacity of the warehouse at Chicago and constructing a new warehouse in Kansas?" The solution is to generate two scenarios, evaluate a performance metric, and print the trade-off between the two alternatives in terms of the performance metric.

Experience

KBS was written in the Schema Representation Language and implemented in Franz Lisp running under the Unix operating system. It is now being reimplemented in Knowledge Craft, a derivative of the Schema Representation Language.[13] Programmers in the Schema Representation Language and KBS envi-

```
                    KBS INTERFACE PROCESS
Expert Run Log of Model INET:

Performing analysis on experiment e0 in context g00039.
Current scenario has rating of -0.3707022476452817.
Attempting to construct better scenario, KBS-experiment-1.
Increasing production rate of M1 from 50 to 92
 because of high stockouts at D1.
Performing analysis on experiment KBS-experiment-1 in context g00278.
Current scenario has rating of -0.3685768787250583.
Attempting to construct better scenario, KBS-experiment-2.
Increasing production rate of M1 from 92 to 134
 because of high stockouts at D1.
Increasing production rate of M2 from 100 to 110
 because of high stockouts at D2.
Performing analysis on experiment KBS-experiment-2 in context g00521.
Analysis failed after 3 iterations.
Best scenario is experiment KBS-experiment-2 with rating = -0.340278656245922

Command: [help] showlog

  MODEL : INET                                    Wed Oct  9 18:17
```

Figure 15. Analyzing hypotheses.

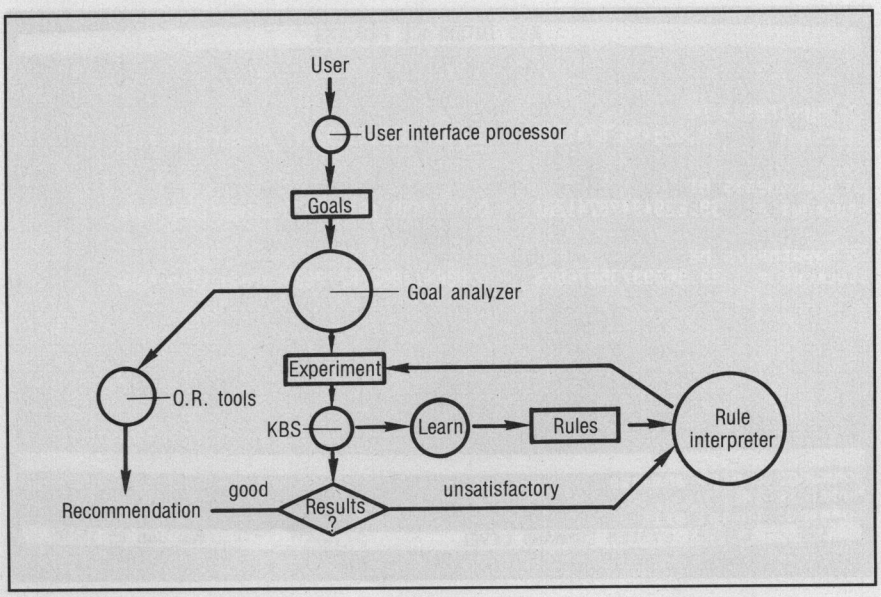


Figure 16. Architecture of a KBS-based expert system.

ronment can also use a rich set of intelligent tools and interchangeably adopt any of the programming styles commonly found in artificial intelligence, such as logic, rule-based, data-driven, and objected-oriented programming.

During the last four years, KBS has been used in several real-world simulation applications, including a printed circuit board manufacturing facility, a light bulb manufacturing facility, a flexible assembly plant, and a corporate distribution and inventory system.

The earliest project, the printed circuit board manufacturing facility, lasted about one man-year. It was a testbed for developing the initial KBS kernel and demonstrating the feasibility of using objects to represent entities and rules. That system boasted an intricate spatial representation of the manufacturing facility using graphics. Managerial personnel who were not domain experts, but who could understand the importance of the performance metrics, greatly benefited from the graphics displays generated by the model.

The light bulb factory and the flexible assembly plant were short experimental ventures (three man-months each) that were a part of other bigger projects.

Since then, most of our experience has been with the corporate distribution and inventory system, called I-Net,[14] which we continue to work on. The I-Net project began with the development of a prototype model with only one business unit (an administrator) serving distribution centers, vendors, resellers, and stocking points.

With this prototype we illustrated in concept the advantages of using KBS as a modeling tool. We animated this model with color graphics, and generated fairly sophisticated reports that used multiple windows to display performance data aimed at different audiences. We also demonstrated voice output.

Throughout the project, all events in the model were designed by domain experts (both programmers and nonprogrammers). It was a tribute to the underlying Schema Representation Language that these people quickly became active participants in the model construction phase as well as in the subsequent model modification phase. The domain experts had no difficulty understanding the model because it directly mirrors the system being modeled.

Encouraged by these results, we increased the level of detail in the representation and event behavior in the early I-Net model until we finally constructed a full-blown realistic I-Net model, called the INET-pc. The specification of event behaviors, which required programming in Lisp, continued to pose difficulties. However, the introduction of event rules in schema form with embedded Lisp functions helped us understanding model dynamics.

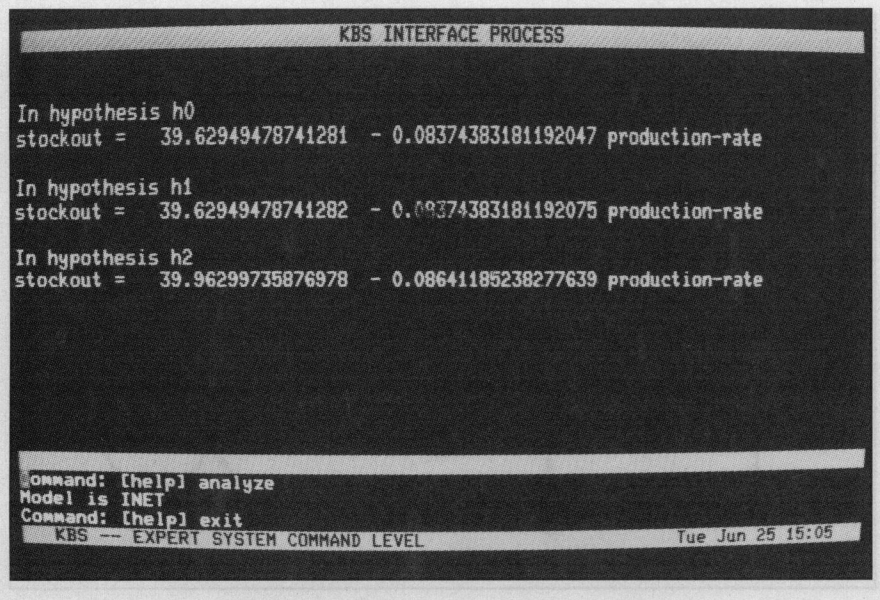


Figure 17. Recommendation from automatic analysis.

Creation of the INET-pc model happened quickly, largely because of a network editor customized to suit the distribution domain. The model validation facilities built into KBS also helped us maintain a correct model at all times. The introspection feature aided debugging, as well as inspiring confidence in the correctness of the model.

It was then that we developed a formal notion of an instrument to collect data during model executions. A natural-language-based command set was also hurriedly developed to peruse the large model, and several business reports using graphics were incorporated to suit the taste of managers.

Unfortunately, with the added functionality and large size of the INET-pc model's network (about 80 facilities), the execution speed of the simulation dropped so much that we were forced to stall development and testing to continue our research into automating simulation analysis. This time, we are using an abstract corporate distribution model such as the one shown in Figure 1.

In this recent phase of the project, we made several important additions to KBS. At a more mundane level, a fairly comprehensive statistical library, which included a causal path analysis package, was created in Franz Lisp. The existence of a context mechanism in the Schema Representation Language was then exploited to simultaneously maintain several model scenarios to let us reason in alternate worlds.

This set up the framework for an expert system architecture where several experiments could be conducted in the same session — with diagnosis and correction between each experiment. Rules for diagnosis are written in OPS5. The working memory elements they operate on are created from model schemata before the rules are fired. This conversion from schemata to working memory elements and back into schemata is made possible by local utilities.

When rules fire, they create specifications for future experiments. The rules we created for our simulations were made only to illustrate our system — they were not handed down to us by the experts. When we do get a set of rules from the experts, they will be quantified with path analysis. Before we do that, we expect to get better hardware and software, thereby improving

the slow execution speeds that discouraged us with the INET-pc model. The I-Net project is now focused on another aspect of the expert system concerned with vehicle dispatching.

Since 1980, KBS has been a testbed for experimenting with knowledge-based techniques for model building, execution, and analysis. As such, it has proved invaluable in identifying user needs and testing approaches to solving those needs.

In retrospect, the first step of using a schema representation and a combined object- and rule-based programming paradigm to represent simulation knowledge and behavior was perhaps the simplest and most intuitive. It made model creation easier because models could be built with objects specific to the domain. It also made understanding the model easier.

But there is a limit to both ease of creating and understanding a model. As models became more complex, simple schema editors weren't sufficient for creation and perusal. Powerful graphics-based model-building facilities were not explored due to the lack of facilities. By combining good graphics and knowledge-based techniques, such as model verification, interfaces may be constructed to support the building of more complex models.

Perhaps the most important aspect of KBS is its focus on automating the simulation life cycle — in particular, its focus on the automated analysis expert system that can conduct experiments and rate scenarios to make a recommendation.

The purpose of simulation is to produce data that, when analyzed, will identify important aspects of the model. Data is often voluminous, and the analyst often lacks requisite skills. By embedding knowledge into KBS to automate the analysis, such expertise can be uniformly and consistently applied across many applications.

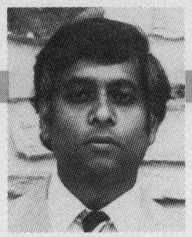
The ultimate goals of applying knowledge-based systems to simulation should be to reduce the total simulation life-cycle time and to increase the quality of the results by making expertise more readily available to the end user. KBS represents a good first step along this path. □

Acknowledgments

The authors thank Donald Butcher and Stanley Wearden for suggesting the path analysis approach to causal analysis. This research was supported, in part, by Digital Equipment Corp., Westinghouse Electric Corp., and Carnegie-Mellon's Robotics Institute.

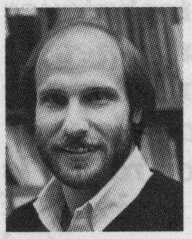
References

1. P. Klahr and W.S. Fought, "Knowledge-Based Simulation," *Proc. First Conf. AAAI*, Stanford, Calif., 1980, pp. 181-183.
2. Mark S. Fox, "The Intelligent Management System: An Overview," Technical Report CMU-RI-TR-81-4, Carnegie-Mellon University, Pittsburgh, 1981.
3. Y.V. Reddy and Mark S. Fox, "KBS: An Artificial Intelligence Approach to Flexible Simulation," Technical Report CMU-RI-TR-82-1, Carnegie-Mellon University, Pittsburgh, 1982.
4. Mark S. Fox and Y.V. Reddy, "Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation," *Proc. 13th Pittsburgh Conf. Modeling and Simulation*, April 1982.
5. Mark S. Fox and Y.V. Reddy, "Knowledge Representation in Organization Modeling and Simulation: A Detailed Example," *Proc. 13th Pittsburgh Conf. Modeling and Simulation*, April 1982.
6. Y.V. Reddy and Mark S. Fox, *KBS: A Knowledge-Based Simulator User's Manual*, Carnegie-Mellon University, Pittsburgh, June 1983.
7. J.M. Wright and Mark S. Fox, *SRL 1.5 User Manual*, Carnegie-Mellon University, Pittsburgh, 1983.
8. Malcolm McRoberts, Mark S. Fox, and Nizwer Husain, "Automating the Analysis of Simulations in KBS," *Proc. SCS Multiconference AI, Graphics, and Simulation*, San Diego, Jan. 1985.
9. Venkataseshan Baskaran and Y.V. Reddy, "An Introspective Environment for Knowledge-Based Simulation," *Proc. Winter Simulation Conf.*, Austin, Texas, 1984.
10. Y.V. Reddy, Mark S. Fox, and Nizwer Husain, "Generating Model Abstraction Scenarios in KBS," *Proc. SCS Multiconference AI, Graphics, and Simulation*, San Diego, Jan. 1985.
11. Mark S. Fox, *Job-Shop Scheduling: A Study of Constraint-Directed Reasoning*, PhD dissertation, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, 1982.
12. C.C. Li, *Path Analysis — A Primer*, Boxwood Press, Pacific Grove, Calif., 1975.
13. *Knowledge Craft*, Carnegie Group, Inc., Pittsburgh, 1985.
14. Y.V. Reddy et al., "I-Net: A Knowledge-Based Simulation Model of a Corporate Distribution System," *Proc. IEEE Conf. Trends and Applications*, Gaithersburg, Md., May 1983.



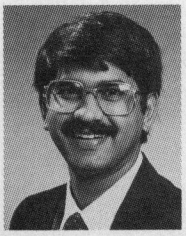
Y.V. Ramana Reddy is a computer science professor and director of the Artificial Intelligence Laboratory at West Virginia University. He is also a visiting professor at Carnegie-Mellon University's Robotics Institute.

Reddy's research interests include artificial intelligence programming environments, knowledge-based simulation, and intelligent decision-support systems.



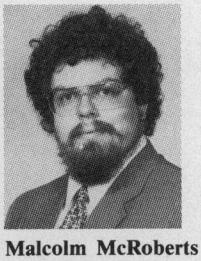
Mark S. Fox heads the Intelligent Systems Laboratory at the Robotics Institute. He is also a founder and director of Carnegie Group, Inc.

Fox received a BS in computer science from the University of Toronto and a PhD in artificial intelligence from Carnegie-Mellon University.



Nizwer Husain is a research programmer at the Intelligent Systems Laboratory. His interests are artificial intelligence techniques applied to management decision making, scheduling, simulation, and graphics.

Husain received an M.Tech in computer science from the Birla Institute of Technology and Science in India and an MS in computer science from West Virginia University.



Malcolm McRoberts is a senior engineer at McDonnell Douglas Astronautics. Before that, he worked at the Intelligent Systems Laboratory.

McRoberts received a BS in applied mathematics and computer science from Carnegie-Mellon University.

The authors may be contacted at the Intelligent Systems Laboratory, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213.