Knowledge Based Simulation: An Artificial Intelligence Approach to System Modeling and Automating the Simulation Life Cycle

Mark S. Fox, Nizwer Husain, Malcolm McRoberts and Y.V.Reddy

CMU-RI-TR-88-5

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

13 April 1988

This research was supported, in part, by Digital Equipment Corporation, Westinghouse Electric Corporation, and the CMU Robotics Institute.

Table of Contents

List of Figures

Abstract

Abstract: This paper summarizes the past eight years of research in the application of Artificial Intelligence to Simulation. Our focus has been in two areas: the use of AI knowledge representation techniques for the modeling of complex systems, and the codification of simulation expertise so that it can be used to manage the simulation life cycle. The KBS system is an embodiment of this research. It provides a complete Simulation Decision Support Environment for the modeling, validation, simulation and analysis of complex systems. KBS has been applied to a variety of problems including factory and distribution system analysis.

1. Introduction

Industry has been slow in adopting simulation as a means for analyzing complex decision problems. One reason is that the complexity of the modeling language, and differences between simulation modeling concepts and the system to be modeled make model building a difficult and time consuming task. Early work in *Knowledge Based Simulation* [Klahr & Fought 80, McArthur & Sowizral 81, Reddy & Fox 82a, Reddy et al. 86] has attempted to remove this problem by using Artificial Intelligence (AI) knowledge representation techniques, such as frames, to represent the objects and their relationships, and rules, to represent the procedural behaviors of the objects[1], to create simulation models which are:

- explicit,
- understandable,
- modifiable, and
- self-explanatory.

By using a frame language to represent domain concepts, such as object structure, and goals, there is a one to one correspondence between the domain and the simulation model[2]. Secondly, by using rules to represent object behavior, the specification and modification of the behaviors become easier. Lastly, explanation techniques developed around rule based systems provide the basis for explaining event behaviors.

While the AI approach has reduced the difficulty of model building, somewhat, more widespread use of simulation technology will not be achieved until the time it takes to perform the activities in the simulation life cycle:

- Problem formulation
- Modeling building
- Data acquisition
- Model translation
- Verification
- Validation
- Experiment planning
- Experimentation, and
- Analysis of results

is reduced, while at the same time the quality of the results are enhanced. The barrier to achieving these goals is the lack of available expertise both in simulation theories and techniques and the domain of application. This lack of expertise results in:

- inaccurate and incomplete models
- poor experiment designs
- poor analysis
- few ideas of how to alter the model to maximize the simulation goals

The representation and utilization of expertise has been one of the more important contributions of AI. Consequently, AI knowledge engineering tools, such as Knowledge Craft[R] [Knowledge Craft 85], provide an excellent environment for constructing a Knowledge Based Simulation Tool for supporting and managing the simulation life cycle and applying expertise at each stage of the cycle.

---

[1]Confusion exists around the use of a knowledge engineering tool to perform simulation and the use of knowledge representation. In the former, the powerful graphic facilities provided by the tool and the underlying workstation provide a rich and powerful interface which does not necessarily have any AI content.

[2]There is another confusion between the concepts of objects such as in Simula/Object Oriented Programming and knowledge representation. Knowledge representation, in addition to being able to represent objects and their procedural behavior [Rychener 82, Knowledge Craft 85], focuses on the relations among objects and the deductions supported by them [fox79]. Even more so, knowledge representation research is directed towards the developement of a clear, concise and consistent semantics for the representation of knowledge. Consequently, standard representations have been developed for a number of domains such as factory scheduling [fox86] and project management [sathi86].

In this chapter, we describe the knowledge based simulation system KBS. KBS has been a testbed for exploring the concept of AI in Simulation. Since 1980, we have been exploring issues such as:

- Using an object oriented approach to model representation where objects have a one to one correspondence with domain objects, and the objects have methods which define their behavior. This provides flexibility in creating and altering entities and their behavior, without altering the simulation model interpreter [Fox & Reddy 82].
- Extending the object oriented representation with concepts from semantic networks in order to provide a standard semantics for representing entities and their relationships [sathi85].
- Representing an object's behavior (i.e., events) in the form of rules which are easily understood by the user [Rychener 82].
- Combining menus and graphics to provide interactive model building.
- Verifying models by using logic programming to specifying verification axioms.
- Providing the capability to reduce a model's complexity through the use of abstraction mechanisms so that a model can be represented at multiple levels of abstraction. The user specifies the level of simulation and the system automatically configures the model [McRoberts, Fox & Husain 85].
- Focusing the gathering and analysis of simulation data according to goals specified by the user [Reddy, Fox & Husain 85, Venkataseshan & Reddy 84].
- Enhancing the user's understanding of a model's dynamic behavior by providing a variety of simulation monitoring facilities including: stepping, tracing, the display of inter-event and intra-event communications, and interrupts and checkpoints to suspend a simulation run to investigate entities or take checkpoints so as to explore the effect of alternate decisions while preserving the option of restoring the system to any one of the several checkpointed model states.
- Rating simulation results according to the goals provided by the user.
- Heuristic (i.e., rule-based) analysis of simulation data and the specification of repairs to the model in order to better satisfy the predefined goals.
- Automatic rule refinement using causal path analysis to determine the degree to which variation of a given effect is determined by each particular cause in the model, achieved by combining the qualitative knowledge regarding causal relations with the quantitative knowledge furnished by correlation and regression.
- Managing the simulation life cycle by means of a goal directed rule system which examines the performance of a scenario with the help of diagnosis/repair rules which can suggest model modifications that may realize a simulation goal, thereby transforming simulations from being descriptive systems to prescriptive systems.

Section 2 describes the distribution domain which will be used as the primary example throughout the article. Section 3 describes the KBS methodologies for representing, creating and verifying models. Section 4 describes goal acquisition, model instrumentation, simulation execution and data gathering. Section 5 describes how KBS learns rule refinements from simulation data. Section 6 describes the overall goal directed architecture of KBS. Lastly, we conclude in section 7.

2. Corporate Distribution Domain

In a large manufacturing organization the corporate distribution system plays an important role in assuring adequate market penetration and retention for its products. This is accomplished by keeping transportation and warehousing costs low,and facilitating an aggressive pricing policy. In addition, it should provide a good level of service to its customers and resellers by providing products on time while not requiring a high level of customer inventory. In order to achieve this the corporate distribution system

should simultaneously deal with a number of mutually conflicting policies. For example, if a reseller is required to carry a high inventory level to minimize stockouts it will reduce the cost of transportation but will exact a penalty on the reseller's ability to be competitive which may result in loss of market share for the corporation. In order to deal with issues such as this a corporate distribution system needs a tool to aid in decision making. Even though many analytical techniques for dealing with multi-echelon inventory systems do exist, we chose a simulation approach so that we can effectively incorporate many idiosyncratic policies that are normally part of complex distribution management systems. The following provides a more detailed description of the distribution system.

A manufacturer produces a number of products which consist of a large number of components and subassemblies some of which are produced by the manufacturer at widely distributed locations while others are purchased from vendors located worldwide. These components and subassemblies are transported to a number of *distribution centers* where they are stocked. Each distribution center serves *customers* located in its assigned area. The customer may be a *reseller* or special customer who can deal with the distribution center or the *corporate business unit* directly. Customer requests are processed by the business unit or the distribution center which results in shipment of products or components to be merged at the customer site. The distribution centers in turn depend on manufacturing centers and vendors to supply components and products to replenish their stock. This problem is further complicated by factors such as seasonal demands, varying lead times to build or expand manufacturing facilities, need to maintain uniform production levels, contractual agreements with vendors, effects of weather and labor problems on transportation schedules and myriad other problems. It can be easily seen that the corporate distribution problem puts tremendous demands on managers at all levels who are faced with decision making which has far reaching effects on the entire corporation.

Consider some of the decisions faced by managers at various levels in the Corporate Distribution System (CDS). The primary objectives of simulating a CDS is to be able answer questions similar to those listed below.

- Where should we locate manufacturing plants for various components and what should their capacities be?
- For a given forecasted demand and its geographic distribution where should we locate distribution centers and what should their capacities be?
- Should the products be merged at distribution centers or at customer sites?
- What is the effect of transportation modes and schedules on customer stockouts and satisfaction?
- What is the overall effect of a delay in vendor shipment of some key components?
- Do we have enough manufacturing and distribution capacity to meet an anticipated increase in demand for products?
- What is the effect of consolidation of manufacturing and distribution facilities?
- What is the effect of a proposed order handling procedure on the corporation?

This list illustrates the enormous complexity of the distribution domain and suggests the need for tools to aid in decision making at a number of levels. For example, low inventories can economize inventory carrying costs but lost sales resulting from frequent stockouts can reduce total profits. Because of this, the tool must inherently be able to deal with conflicting goals. The KBS approach to simulation will make it possible to deal with such issues.

A simplified model of a Corporate Distribution network is shown in Figure 2-1. (See [Reddy et al. 83] for the earlier work upon which this model is derived.)

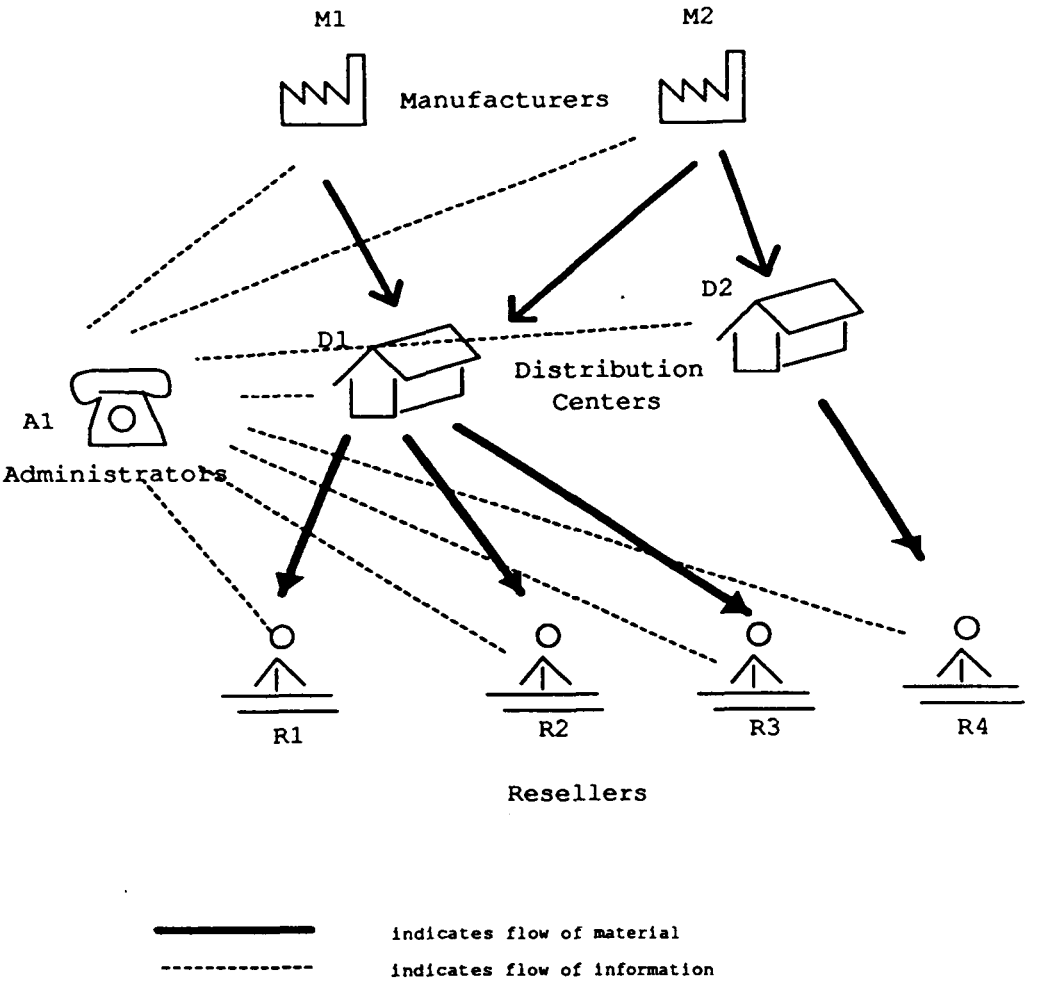

Figure 2-1: Example of a Simplified CDS Model

3. Model Building

The acquisition, representation and verification of simulation models probably consumes the majority of the time spent in the simulation life cycle. The following describes how we have applied AI to increase the quality of models and decrease the time it takes to build them.

3.1. Model Representation

It is our belief that the greater the cognitive distance between an expert's description of a system and the descriptions directly supported by a simulation language, the greater the difficulty there is in constructing the simulation model. Consequently, we have strived in KBS to support the construction of models which closely match the way the expert views the system.

The approach taken in KBS to represent models is based on AI knowledge representation techniques [Reddy & Fox 82a, Fox & Reddy 82, Reddy & Fox 82b, Reddy & Fox 83]. The representation is a frame based semantic network, which supports the representation of objects, their attributes, structure and procedural behaviors, relations among objects, goals, and constraints. Models are constructed by first

building a set of prototypical objects which are standard within a particular domain. In the case of distribution management, some standard objects are resellers, distribution centers, and manufacturing sites. A specific model is built by instantiating these prototypes and connecting them via relations.

Objects in KBS are represented as **schemata**[3]. A schema is represented as in Figure 3-1, with opening double braces followed by a schema name (printed in bold font) and a set of slot-value pairs,and finally terminated by closing double braces.

The **pittsburgh-reseller** schema defines a retail sales store containing two stock items a and b. It also defines who the manager is, its sales region, and its address.

```
{{ pittsburgh-reseller
        INSTANCE: Reseller
        STOCK: stock-a stock-b
              range: (type instance stock)
        REGION: Allegheny
        MANAGER: Ramana Reddy
        ADDRESS: 123 Easy St.
        REORDER-PROCEDURE: Reorder-rule-1
```

Figure 3-1: Example of a Reseller schema

In addition to values, each slot may have a set of associated facets or **meta-information** (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. The range facet restricts the types of values taken on by a slot.

The **stock-a** schema defines the status of the in stock part a and what has been ordered due to the part falling below its reorder point.

```
{{ stock-a
        INSTANCE: stock
        PART: a
        ON-HAND: 138
        ON-ORDER: {{ INSTANCE: stock-order
                        PART: a
                        AMOUNT: 200
                        SOURCE: pittsburgh
                        DATE-ORDERED: 1 aug 87
                        DATE-DUE: 15 aug 87 }}
        MIN-STOCK: 200
        RE-ORDER-AMOUNT: 500 }}
```

An important aspect of CRL 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 along these relations. For example, **pittsburgh-reseller** is related to **reseller** by the INSTANCE relation. It inherits its standard slots from **reseller**, but their values are defined locally.

The procedural behaviors of an object, such as the reorder procedures for pittsburgh-reseller, are defined by a slot which names the procedure, and by the values of the slots which define the actual

[3]Early versions of KBS were built on top of SRL [Wright & Fox 83], a knowledge engineering tool, subsequent versions have been implemented in Knowledge Craft's representation language CRL [Knowledge Craft 85].

behavior. Values can be lisp procedures or rules. A rule provides a means of specifying procedural knowledge which is easier to comprehend by the model builder. For example, the REORDER-PROCEDURE for the **Pittsburgh-Reseller** has its behavior defined by the following rule:

```
{{ Reorder-rule-1
    INSTANCE: rule
    IF: (less-than stock.on-hand stock.min-on-hand)
    THEN: (send-message :to stock.source
                        :message reorder
                        :part stock.part
                        :amount stock.reorder-amount)
          (create-reorder-record stock) }}
```

The rules reorders a part when the on-hand stock is below the reorder point by sending a message to the object which is the source of the part for the reseller.

Knowledge Craft provides the model builder with a graphical interface for defining new schemata and slots (figure 3-2), and to define the inheritance semantics of slots which act as relations. This includes defining what information (slots and their values) is inherited, not inherited, and altered when inherited. This feature will be useful in establishing special relationships between modelling entities that deal with restricted access to information and automatic elaboration of requested information. For a comprehensive treatment of this concept consult [Knowledge Craft 85].

3.2. Model Acquisition

Two philosophies exist for the acqusition of simulation models: a domain independent view where the interface utilizes simulation concepts as the constructs to be described, versus a domain dependent view where domain specific concepts are the constructs to be described. Experience has shown [Kahn 87] that domain specific knowledge acquisition systems provide a powerful means of acquiring models.

The philosophy taken in our work on knowledge based simulation has been to develop a set of domain specific interfaces. With the availability of powerful knowledge engineering tools on workstations with good graphic displays, the door has been opened for the development of rich interfaces. For example, a descendent of the our KBS work at CMU is the Simulation Craft system developed at Carnegie Group [Fox et al. 86]. In figure 3-3 there is an example of a multiwindow interface which contains a window for process planning and another for facility layout.

3.3. Consistency and Completeness

A recurring problem in simulation systems, including KBS, is maintaining model *consistency* and *completeness*. We found that much time is wasted discovering errors and holes in the model. To deal with this problem, we use the logic programming facility of Knowledge Craft to specify completeness and consistency rules.

A consistency constraint relating resellers and distribution-centers may be specified as:

```
(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.

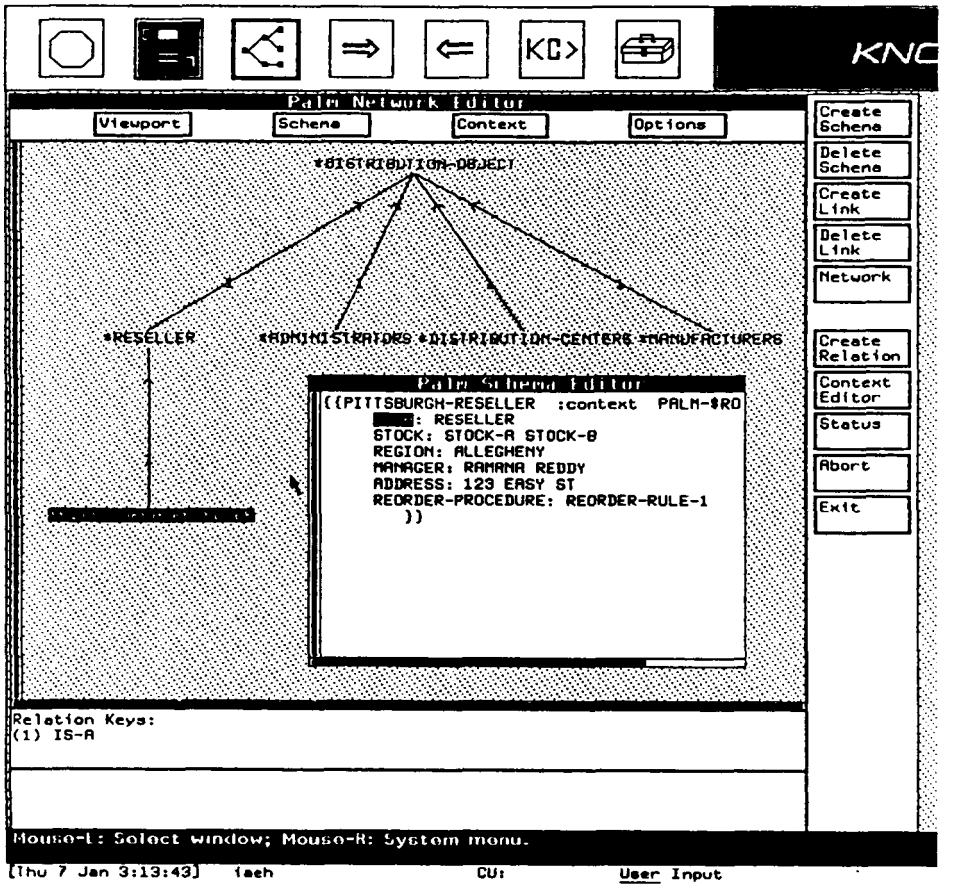

Figure 3-2: The Knowledge Craft Representation Building Interface

For each constraint, KBS evaluates it and reports whether it was satisfied or whether it failed. In case of a constraint failure the interpreter provides a trace facility to determine the source of failure.

3.4. Model Reduction

The model we developed of a corporate distribution system was large in terms of the number of facilities, and complex in terms of decision processes and levels of detail. In working with the distribution analysts, we found that the entire model was not necessary to answer every question. Instead, a question required a version of the model which is reduced in either breadth or depth. In particular, when focusing on a particular region's distribution logic, only abstractions of other regions were required. Consequently, it was necessary to introduce a mechanism for abstracting selective portions of the model. These alterations can be performed automatically provided the model builder has created a knowledge framework for the task. These simplifications result in faster running models, increased model understanding, and simplified analysis.

Model simplification techniques fall into two main categories, static and dynamic. In static techniques both model structure and model parameters are altered; however, the event behaviors remain unchanged. Since only static aspects of the model are affected these are called static techniques. Dynamic techniques on the other hand alter a model's dynamic processes. This means redefining some of the event behaviors.

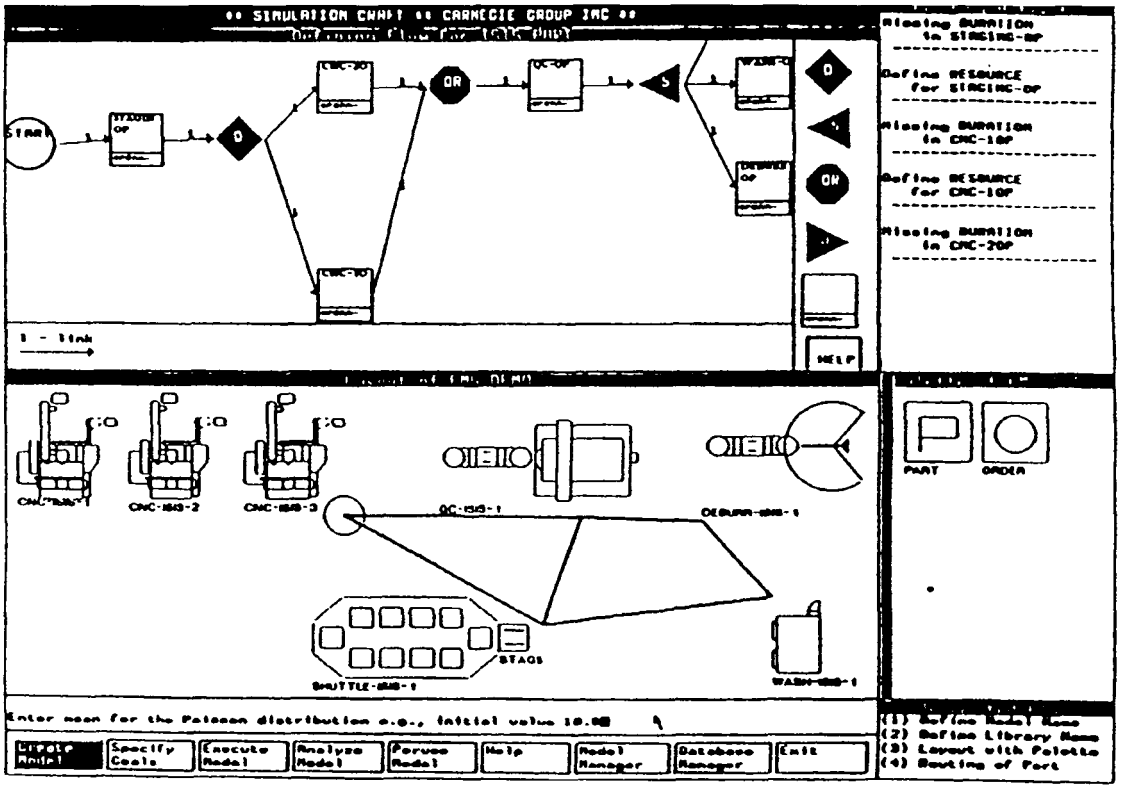

Figure 3-3: Process plan and facility layout interface

3.4.1. Static Abstraction

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. This will result in information that is unique to the individual nodes being lost but if the new node's parameters are adjusted correctly then the new node will be functionally similar to the old group. In this manner the rest of the model is not affected. For example, in Figure 3-4 individual resellers in a region may be grouped together to form a regional reseller. The corresponding schema that describe this kind of aggregation is shown in Figure 3-5.

{{ **equivalent-node-aggregation**
IS-A: model-abstraction
TYPE: reseller
comment: type of node to aggregate
SUPER-NODE: regional-reseller
comment: single node replaces sub-nodes when abstracted
SUB-NODES: reseller1 reseller2 reseller3
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 3-5: Example of Equivalent Node Aggregation

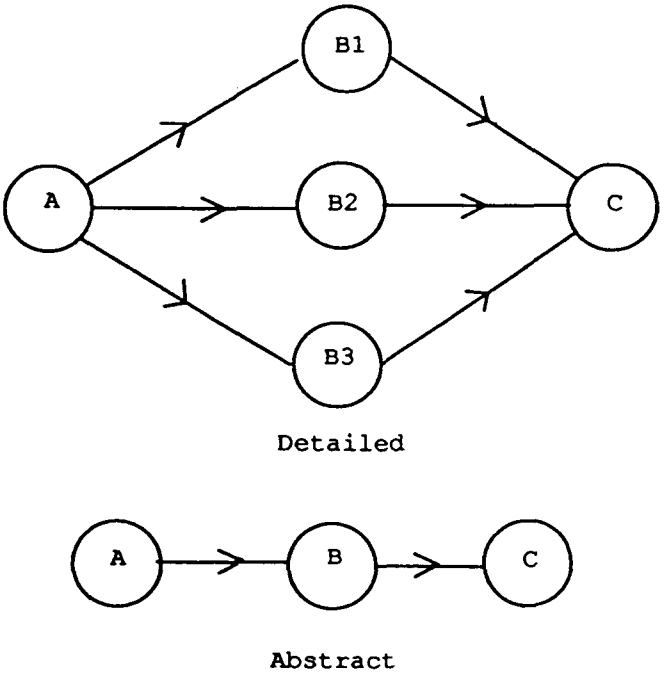

Figure 3-4: Equivalent Node Aggregation

In data class aggregation objects are grouped into classes and all references to the members are replaced by references to the class. This of course throws away information about variations among the class members but it should increase the overall efficiency and not greatly affect model execution.

An example of this concept as applied to the distribution model would be to group all items sold into classes. For instance in the computer business different types of personal computers may be grouped together into a broad class referred to as PC. This would greatly reduce the amount of data needed to track inventory levels for each separate type of personal computer in the model.

3.4.2. Dynamic Abstraction

Dynamic abstraction attempts to simplify simulation models by analyzing the stimulus-response event behavior of one or more nodes, and to construct a single node with a stimulus-response behavior which is statistically similar. This technique can be used to combine nodes of the same or different classes. The collection of statistics on event behavior can be achieved by either constructing stimulus response frames, or by post-processing information gleaned from model introspection. However purely statistical information on events is difficult to use because event parameters cannot readily be asbtracted, and in such cases the abstracted model may become inconsistent. A full treatment of abstraction techniques as used in KBS is provided in [McRoberts, Fox & Husain 85].

4. Model Simulation

In KBS, simulation is viewed as being composed of a sequence of *experiments*, where each experiment measures how well a *scenario* (i.e., an altered version of the original model) optimizes one or more *goals*. The process of simulation begins first with the specification of a set of goals which result in the instrumentation of a scenario in order to gather data. The scenario is then executed interactively with an animated display. The scenario is then rated as to how well it has optimized the goals. The rest of this section provides a detailed description of the process.

4.1. Goal Specification and Instrumentation

The primary purpose for constructing a simulation is to verify a hypothesis or optimize one or more features of a system. Optimization can occur if we are able to measure the performance of a scenario. We introduce the concept of rating a scenario as a method of measuring the goodness or badness of simulation results. For example, if the goal is to utilize machines to their maximum (100%), and if in a given simulation the average utilization of machines is 90% and when confronted with the question "have we reached the goal ?", we would like to give a better answer than **no**. To rate scenarios more smoothly we chose a continuous scale of rating from -1 to +1, -1 meaning the results are far from the goal and +1 to indicate the goal has been completely satisfied. Since goals are often complex and may be composed of conflicting sub-goals, we therefore describe here an approach to specification of goals of a simulation as a composite of a set of constraints[4] on the performance of various entities of the system being modeled.

Following are the steps involved in the construction and evaluation of goals.

- Represent each organizational *goal* as composed of a set of *constraints*.
- Select and attach *instruments* to gather data.
- Specify procedures for computing the performance measures from the raw data collected.
- Execute the simulation for the given scenario.
- Evaluate each constraint by computing a coefficient of constraint satisfaction, which may be positive to indicate reaching the desired goal, or negative, indicating falling short of the desired goal.
- Evaluate the scenario by computing a coefficient of goal satisfaction as a weighted average of constraint satisfaction coefficients.

4.1.1. Goal Representation

A goal is an aggregation of subgoals which we call *constraints*. A goal is evaluated as a weighted sum of the individual subgoal constraint satisfaction coefficients. Figure 4-1 shows an example goal which when evaluated yields a RATING which is a measure of the desirability of the given scenario. This rating is derived by combining the ratings of individual CONTRIBUTING-CONSTRAINTS of the goal:

- order-fill-rate
- inventory-turns
- inventory-investment
- order-cycle-time
- distribution-cost

Other slots in the goal schema specify the procedure for evaluation and display of the results. Figure 4-2 shows the representation of a constraint. The slot CONSTRAINED-BY of the **constraint** schema contains the details of the instrumentation needed to evaluate a constraint.

---

[4]Constraints are an inherent part of all organizational models. For a detailed treatment of contraints refer to Fox [Fox 83].

{{**inet-pc-goal1**
INSTANCE: KBS-goal
RATING: 0.35
comment: a goodness or badness indicator
CONTRIBUTING-CONSTRAINTS: order-fill-rate inventory-turns
inventory-investment order-cycle-time distribution-cost
comment: the individual goal constraints
STATUS: inactive
comment: whether active or inactive
GRAPH: goal-report-kiviat
comment: a kiviat graph displays rating
EVALUATION-SCHEDULE: every-quarter
comment: governs when to evaluate goal
GOAL-SCHEDULER: scheduler
comment: interprets evaluation schedule
EVAL-FN: eval-KBS-goal
comment: goal evaluation function
REPORT-FN: display-evaluated-goal
comment: function to display goal state }}

Figure 4-1: Composite Goal Schema

{{ **order-fill-rate**
INSTANCE: goal-constraint
CONSTRAINED-BY: order-fill-rate-spec
comment: specification for constraint
CONTEXT: order-fill-rate-precondition
comment: decides if constraint applies
IMPORTANCE: 0.25
comment: relative importance of constraint
RATING: 0.8
comment: a goodness or badness indicator
VALUE: 40000
comment: unrated raw value of constraint }}

Figure 4-2: Constraint Schema

{{ **order-fill-rate-spec**
INSTANCE: goal-constraint-spec
APPLY: interpolate-linear-graph
CONSTRAINT-SPEC-OF: order-fill-rate
UTILITY-GRAPH: (100 -1.0) (10000 0.0) (60000 1.0)
INSTRUMENTS: measure-orders-filled }}

Figure 4-3: Constraint Schema

4.1.2. Instruments

The task of data collection is concerned with recording the changes in the value of a parameter. This can be accomplished by constantly monitoring the parameter and recording every change or by sampling. The former yields greater accuracy albeit with greater computational overhead whereas the latter

approach may be satisfactory in many cases and thus the selection of the data collection method is subjective. In the KBS environment the monitoring is accomplished by attaching **demons** to slots whereas sampling is done by scheduling data collection events or as a separate action during the execution of regular events. Since the data collection in KBS is analogous to physical measurements (using measuring instruments) the notion of an *Instrument* (figure 4-4) is introduced.

An instrument in KBS may be viewed as a probe attached to a schema representing an entity in the model and collects specified data whenever that schema is acted upon in some way. For example, if we are interested in studying inventory levels we may attach an inventory measurement instrument which is activated whenever inventory levels change (figure 4-5). Whenever the instrument is activated it may simply record the value of the simulation clock and the inventory level. This information is stored within the instrument itself (figure 4-6) and can be formatted in a variety of ways using a *report-generator* associated with the instrument. For example if we are interested in the time dependent behavior of inventory we may subject the data collected to an analysis by a *time-series-analyzer*. On the other hand if we are only interested in the minimum inventory, we can subject the same data to analysis by a *descriptive-statistics-analyzer*.

{{ **KBS-Instrument**
IS-A: instrument
PURPOSE:
INSTRUMENT-TYPE:
Range: (OR data-collection data-display)
INSTRUMENT-MODE:
Restriction: (OR demon event scheduled)
SLOT-TO-DEPOSIT:
Comment: slot value to be monitored or the event slot when this will be executed as one of the event actions
SCHEMA-TYPE:
Comment: the names of generic schemata to which this instrument applies
INSTRUMENT-SCHEDULE:
Comment: event schedule for scheduled instrument
Restriction: (TYPE is-a event-schedule)
DATA:
ACTION:
Comment: the data collection or display function
ATTACHMENT-FUNCTION:
Comment: the attachment procedure }}

Figure 4-4: KBS-Instrument Schema

Once an instrument schema is defined, it is attached to an appropriate part of the model, which when executed, results in the collection of the specified data which can be subjected to analysis.

{{ **xa50-inventory-instrument**
INSTANCE: KBS-instrument
INSTRUMENT-TYPE: data-collection
INSTRUMENT-MODE: demon
SLOT-TO-DEPOSIT: on-hand
SCHEMA-TYPE:xa50-inventory
DATA: xa50-inventory-on-hand-data }}

Figure 4-5: Inventory Instrument Schema

{{ **xa50-inventory-on-hand-data**
INSTANCE: set-data
SCHEMA-SET: set of xa50 inventory schemata derived from the instrument
SLOT: on-hand
ANALYSIS: descriptive-stats
DATA: to be filled by the instrument }}

Figure 4-6: Inventory Data schema

The specification of the instrument and data schemata is to be derived manually. However, it may be possible to automate this process in the future by using a natural language deductive reasoning system.

4.1.3. An Example of Scenario Rating

In this section we will illustrate the procedure for rating a scenario via an example from the CDS model. Consider a composite organizational goal to increase customer satisfaction while keeping the distribution overheads low. This goal may be broken-down into two subgoals:

- Customer stockouts should not exceed 5% of orders.
- Distribution cost per unit sold should not exceed 10% of the cost of manufacturing.

{{ **inet-pc-goal1**
INSTANCE: KBS-goal
CONTRIBUTING-CONSTRAINTS: satisfy-customer economize-distribution
EVALUATION-SCHEDULE: daily-at-midnight }}

Figure 4-7: An example of CDS's goal

The corporate goal specified by the schema: **inet-pc-goal1** is a composite of two organizational goal constraints: **satisfy-customer** and **economize-distribution**. These are shown in figures 4-8 and 4-9 respectively. Each goal constraint is assigned an importance rating based on the role it plays in the overall corporate plan. Each goal constraint schema points to a constraint specification schema which specifies a utility function and the needed data collection instruments. Figures 4-10 and 4-11 show the constraint specifications for the **satisfy-customer** and **economize-distribution** constraints.

{{ **satisfy-customer**
INSTANCE: goal-constraint
CONSTRAINED-BY: satisfy-customer-spec
IMPORTANCE: 0.70 }}

Figure 4-8: Retailer Satisfaction Goal Constraint

{{ **economize-distribution**
INSTANCE: goal-constraint
IMPORTANCE: 0.30
CONTEXT: economize-distribution-precon
CONSTRAINED-BY: economize-distribution-spec }}

Figure 4-9: Distribution Cost Reduction Goal Constraint

{{ **satisfy-customer-spec**
INSTANCE: goal-constraint-spec
APPLY: eval-customer-satisfaction
UTILITY: stockouts-utility-graph
INSTRUMENTS: measure-stockouts measure-total-orders }}

Figure 4-10: Specs for Customer Satisfaction Constraint

{{ **economize-distribution-spec**
INSTANCE: goal-constraint-spec
APPLY: eval-distribution-costs
UTILITY: distribution-utility-graph
INSTRUMENTS: measure-manf-cost measure-distribution-costs }}

Figure 4-11: Specs for Distribution Goal Constraint

In order to evaluate the goal constraint: **satisfy-customer** we need to measure the total orders filled as well as the number of stockouts. This is accomplished by depositing the instruments: **measure-stockouts** and **measure-total-orders** shown in figures 4-12 and 4-13 respectively.

{{ **measure-stockouts**
INSTANCE: KBS-instrument
INSTRUMENT-TYPE: data-collection
INSTRUMENT-MODE: event
ACTION: extract-stockout-info
SLOT-TO-DEPOSIT: back-orders
SCHEMA-TYPE: }}

Figure 4-12: Instrument to measure Stockouts

{{ **measure-total-orders**
INSTANCE: KBS-instrument
INSTRUMENT-TYPE: data-collection
INSTRUMENT-MODE: event
ACTION: sum-up-total-orders }}

Figure 4-13: Instrument to measure Total Orders

The data needed to evaluate the **economize-distribution** goal constraint is collected by depositing the instruments: **measure-distribution-cost** and **measure-manf-cost** shown in figures 4-14 and 4-15 respectively.

{{ **measure-distribution-costs**
INSTANCE: KBS-instrument
INSTRUMENT-TYPE: data-collection
INSTRUMENT-MODE: event
ACTION: compute-distribution-costs }}

Figure 4-14: Instrument to measure Distribution Costs

{{ **measure-manf-costs**
INSTANCE: KBS-instrument
INSTRUMENT-TYPE: data-collection
INSTRUMENT-MODE: event
ACTION: compute-manf-costs }}

Figure 4-15: Instrument to measure Manufacturing Costs

The instrument action functions will be tailored to extract the desired information and update the "data" in the instrument. Note that instrument data may also be directly available in the schema it is attached to.

Having defined the goal, its constraints and the instruments, it is then connected to the model. When a goal is connected the relevant instruments are automatically attached to the model. The actual evaluation of the goal is an event scheduled to occur at some future date according to the EVALUATION-SCHEDULE, at which time the EVAL-FN is executed. However, it may also be evaluated manually at any time as desired.

A goal evaluation function (e.g. eval-KBS-goal) will normally retrieve the CONTRIBUTING-CONSTRAINTS of the organizational goal. For each constraint if the **context** of the **constraint** applies then the constraint is evaluated by the function in the APPLY slot 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 it should be possible to observe the direction and decide how good or how bad the organization is doing.

4.2. Experiment Specification

Execution of the model consists of simulation runs. Each run of a model is an experiment. KBS defines schemata which contain all the information needed to conduct a series of experiments on models. At the end of each experiment, KBS may optionally "fire" a rule base which may suggest changes that need to be made to the model in the next experiment.

In Figure 4-16 **KBS-run-spec** specifies the profile for a simple run. RUN-DISPLAY governs what is going to be shown on screen during model execution. If the user is interested in the execution trace of model EVENT-TRACE is activated, and in addition, if system statistics such as events per second of CPU time are needed SYSTEM-STAT is turned on. START-TIME and STOP-TIME refer to the simulation clock. Note, if there are no events in the calendar or if a terminating condition is met, the experiment may stop before the clock reaches STOP-TIME.

{{ **KBS-run-spec**
RUN-DISPLAY: kbs-run-display
comment: the display which is active during the run
EVENT-TRACE: t
comment: if event trace is posted to the display
SYSTEM-STATS: t
comment: if system stats are needed
SYSTEM-STAT-FREQUENCY: 20
comment: tells how often system statistics are updated
START-TIME: "12 Sept 1985"
comment: start time for each experiment
STOP-TIME: "25 Sept 1985"
comment: stop time for each experiment }}

Figure 4-16: The Run specification

Figure 4-17 shows the execution profile needed to conduct a series of experiments. It specifies options for model introspection (LEARNING-REQUESTED), goal-directed scenario rating (RATING-REQUESTED), automatic diagnosis and correction of model parameters through rule-based analysis (RULES-REQUESTED, RULE-SET, ITERATION-LIMIT) and management of related experiments (INITIALIZATION, EXPERIMENTS-TO-BE-DONE, COMPLETED-EXPERIMENTS, BASE-CONTEXT).

{{ **KBS-expert-run-spec**
INITIALIZATION: initialize-run
LEARNING-REQUESTED: no
comment: if introspection is requested
range: (or yes no)
RATING-REQUESTED: no
comment: if scenario rating is to be done, implies goals need to be connected
range: (or yes no)
LEARNED-DATABASE: nil
comment: Name of file to save learned information
RULES-REQUESTED: yes
comment: if automatic diagnosis is required
range: (or yes no)
EXPERIMENTS-TO-BE-DONE: e4
restriction: (TYPE instance KBS-experiment)
COMPLETED-EXPERIMENTS: e1 e2
RULE-SET: bottleneck-diagnosis
comment: Specifies Rule set to use in diagnosis
ITERATION-LIMIT: 8
comment: Number of experiments before the expert system gives up
CURRENT-ITERATION: 3
comment: the current run number
BASE-CONTEXT: kbs-start-context
comment: context where the base model resides,
experimental contexts are children of it }}

Figure 4-17: Expert System Run spec

The **KBS-experiment** schema in Figure 4-18 describes individual experiments. The slot MODEL-CHANGES is filled with a set of "change specs", which describe the changes that need to be performed on the model before the start of each experiment. In addition there are slots (not shown)

whose values reflect the status of that experiment.

```
{{ e3
     INSTANCE: KBS-experiment
     EXPERIMENTAL-CONTEXT: kbs-experiment-context3
          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
     .....
     ITERATION: 3
          comment: the iteration this experiment corresponds to  }}
```

Figure 4-18: Specification of an Experiment

KBS supports a feature rarely found in traditional simulation environments. It is the ability to conduct a series of experiments without human intervention. The important point to note is that individual experiments are not disjoint but are improvements on past experiments, where the improvements are automatically achieved by rule-bases detailing the diagnosis/correction heuristics. In order to work correctly the rules must allow for reasoning between several model scenarios. Management of alternate scenarios in KBS is possible by creating a context tree where each context is devoted to a scenario.

4.3. Simulation Execution

In discrete event simulation the system changes state at discrete points in time. Running the model simply consists of executing the next imminent event from the calendar of events until there are no more events to execute or some halting criteria are satisfied. Each event notice in the calendar specifies the the name of the event, its focus, the time and any parameters that should be available at the execution time. In the example event notice, (see figure 4-19)

```
{{ event24
     INSTANCE: event-notice
     FOCUS   : pittsburgh-reseller
          comment: Focus of event, the entity
     EVENT   : order-arrival
          comment: event-slot in event-schema
     TIME    : "21 April 1985 11:00:00"
          comment: Time of execution
     PRE-ACTION: nil
          comment: Action to take before event execution
     POST-ACTION: nil
          comment: Action to take after event execution
     EVENT-PARAMETER: order10
          comment: Event parameters
     RUN-EVENT: run-event
          comment: method to execute event   }}
```

Figure 4-19: An example Event Notice

an event "order-arrival" is to occur on "21 April 1985" at 11:00:00 AM simulated time and is focused around **pittsburgh-reseller**. The execution of this event is accomplished by the following steps:

- the simulation clock is advanced to "21 April 1985 11:00:00"
- the value of the slot ORDER-ARRIVAL of the schema **pittsburgh-reseller** is extracted

- each item found in the list of values is interpreted

The items found in the list of values in the event slot may be representing a LISP function, a *rule* or an *instrument* which are designed to collect data, display some text or produce graphic side-effects. Simulation proceeds by successively executing event notices until a prespecified terminating condition is met.

4.3.1. Checkpoints and Interrupts

Checkpoints are used in KBS to save the state of a model *during the experiment* along with run-time system information such as the simulation clock, the calendar and other global variable bindings. Checkpoints are also implemented through the use of contexts. Before the experiment is initialized a new context *kbs-run-context-n* which is a child of the experiment context is created to protect that experiment from getting "corrupted" by subsequent modifications. After the "prime event" (i.e. the event that sets the simulation in motion) another context *kbs-run-context-n+1* a child of *kbs-run-context-n* is spawned. In addition, the user may interrupt the simulation any number of times to **checkpoint** the model. This facility allows the user to backtrack to some previous point in execution to compare the differences between progressive scenarios. We refer to checkpoints as snapshots of models *within* the same experiment. Checkpoints are useful when comparing different scenarios in a single experiment.

4.3.2. Tracing

A simulation analyst often follows hunches when debugging simulation models. This style of debugging is different from stepping instruction by instruction, which can be frustrating if, for instance, the analyst is only interested in events focused around *pittsburgh-reseller*. In tracing with graphics, the user fills in a **trace-spec** and the system traces only under conditions declared in the schema. For example, it is possible to selectively observe *pittsburgh-reseller: order-arrival* when *scheduling*, in which case all other events are deemed uninteresting and will not feature in the trace. It is also possible by default to trace everything.

In Figure 4-20 we show a photograph displaying the event to event interaction in the CDS model. The rectangle in the center shows the focus of the current event, "westfield-manufacturing" scheduling two other events focused around "lebc" and "transportation". In practice, we found this to be a powerful feature because it makes the modeler understand the internal workings of the model thus improving one's confidence in the results of the model.

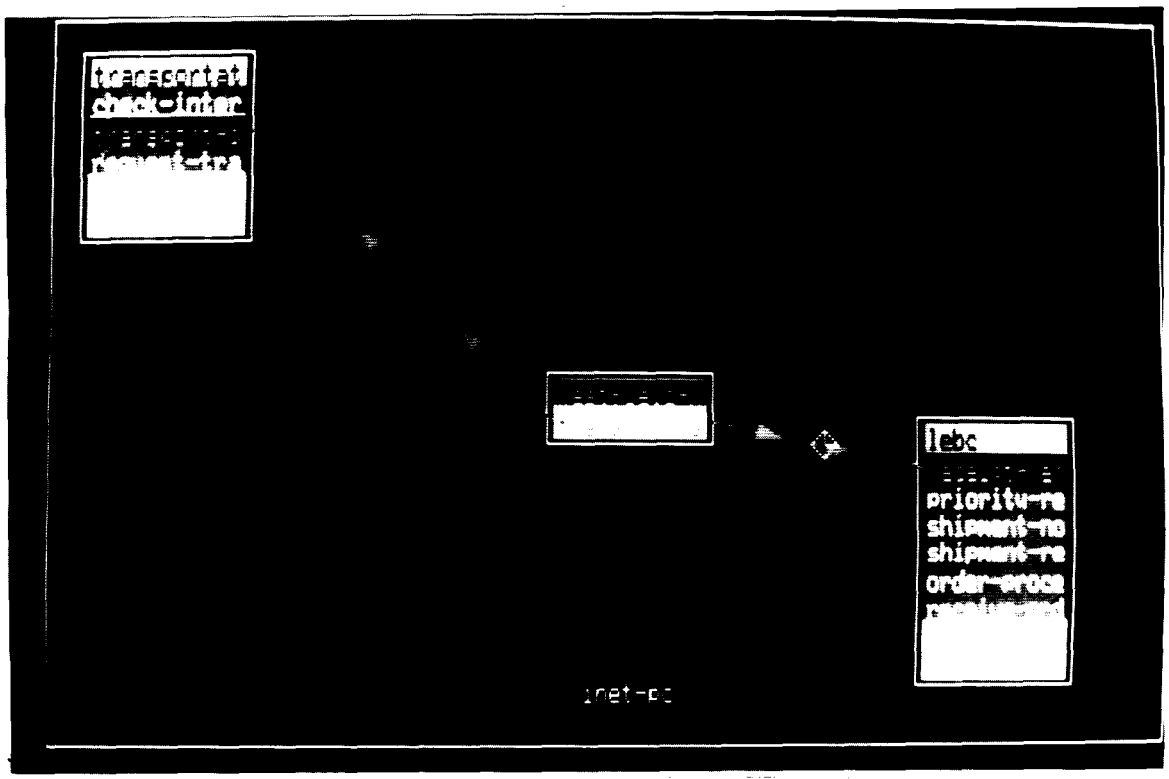

Figure 4-20: Tracing with Graphics in actual runs

4.3.3. Stepping

In tracing we saw how events are displayed while they execute. However, due to the limitations of the screen or due to the unavailability of a graphics device, it is useful to have a *stepping* capability. As events are executed they change values of attributes and such state changes are observed step by step in a window dedicated to stepping. A *step* command is all that is needed to switch on this kind of trace. Stepping is similar to tracing everything, but, without the use of graphics. Attribute value bindings are also shown here because information is displayed textually. To summarize the difference between tracing and stepping, it is easier to detect event attribute chains in tracing, but it is not possible to observe individual event parameters which is better done by stepping.

4.3.4. Event Tracking

The automatic analysis of simulation data (section 5) requires the recording of event sequences and their interaction with entities and their attributes.

For example, figure 4-21 is an example of an event track which includes events E_i and attributes X_j. If we declare that X_1 *causes* E_1 because E_1 *depends-on* X_1 then the system can automatically deduce that

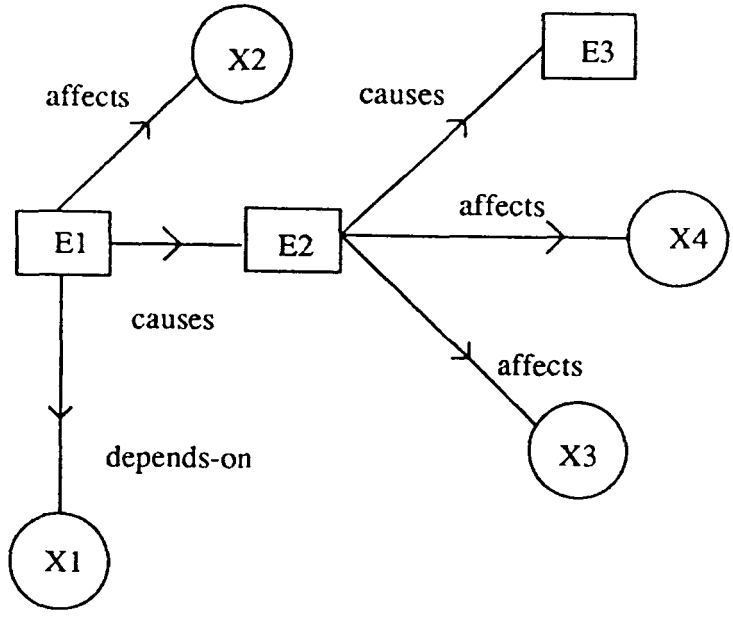

Figure 4-21: Event Tracks

X_1 indirectly affects X_4 thus tracing a causal path from X_1 to X_4[5].

Event tracks are stored as meta information attached to slots which define event behaviors and attributes which are accessed or affected by events.

4.3.5. Simulation Animation

Graphic animation is sometimes the most effective way to represent the dynamics of a simulation. In very large models the screen may not be large enough to display all the information at once to all audiences. In such cases several views of the model may be presented just as is done in perusal. Here, in addition to intelligent use of changing icons and colors, special effects such as flashing and life-like movement may be shown.

[5]There is an important limitation in the automatic detection of cause-effect chains when the attributes are included. The system can only detect how an event affects an attribute and not vice-versa. For example, the system is capable of detecting the fact that the "inventory" is reduced, when the event "sell-goods" is executed. But another fact that the reduction in "inventory" has the effect of causing the event "order-for-goods" goes unnoticed. However since the value of "inventory" is *accessed* during the event "order-for-goods" therefore, for automatic tracing we may make the assumption that *accesses* implies *depends-on*.

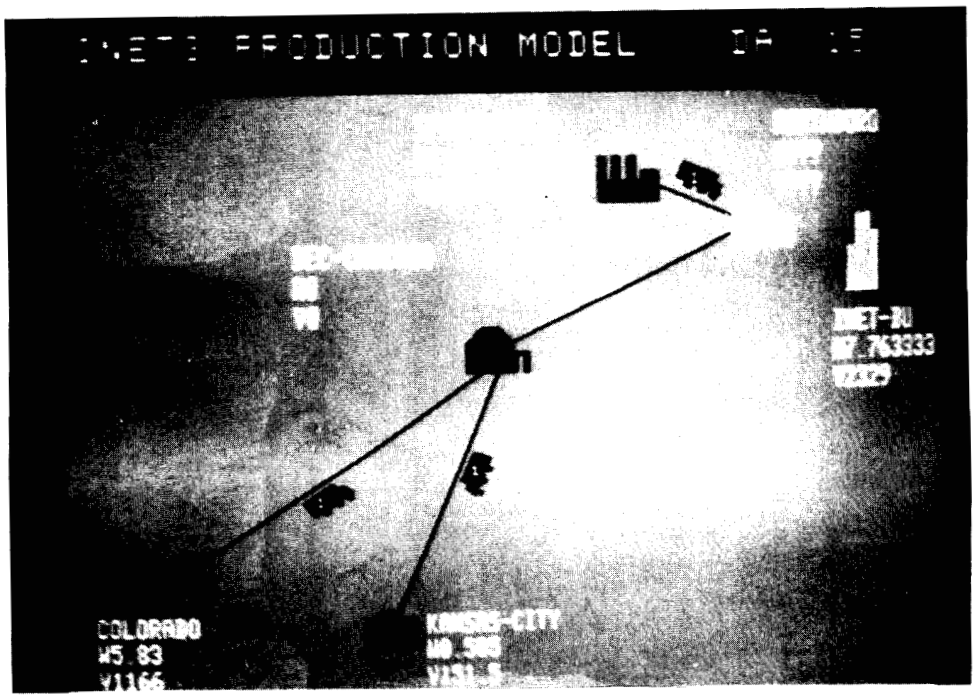

Figure 4-22: Animation of the inet-pc model

Figure 4-22 shows the use of changing colors of icons and animating can be used to effectively communicate the performance of a model scenario.

4.3.6. Report generation

In KBS, performance data is collected by "instrumenting" the model. The data thus collected is stored in the instrument itself or in some other schema. This data has to be summarized and presented before it can be of any use. These summary reports themselves can be generated by using "Report Instruments" that are attached to event slots. As the events are executed, appropriate reports also appear as those instruments are executed. This can take the form of a "textual report" or "graphic side effect" which updates the display sceeen.

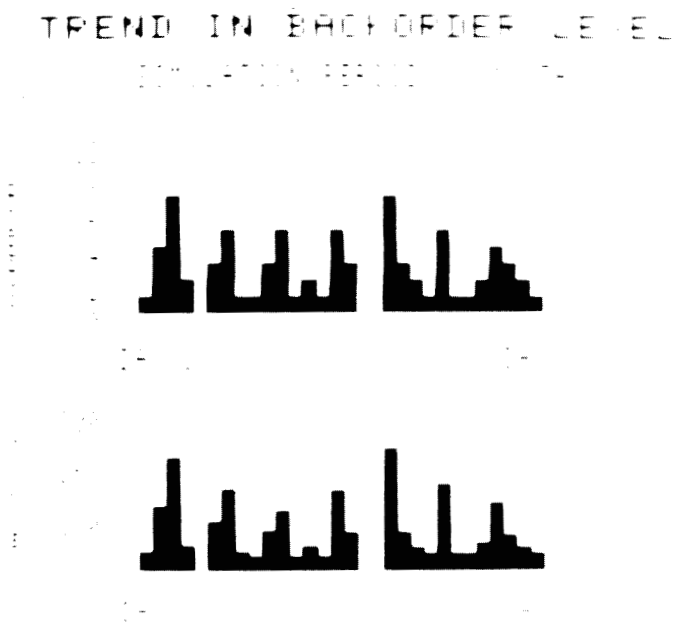

Figure 4-23: Example of Business Graphics

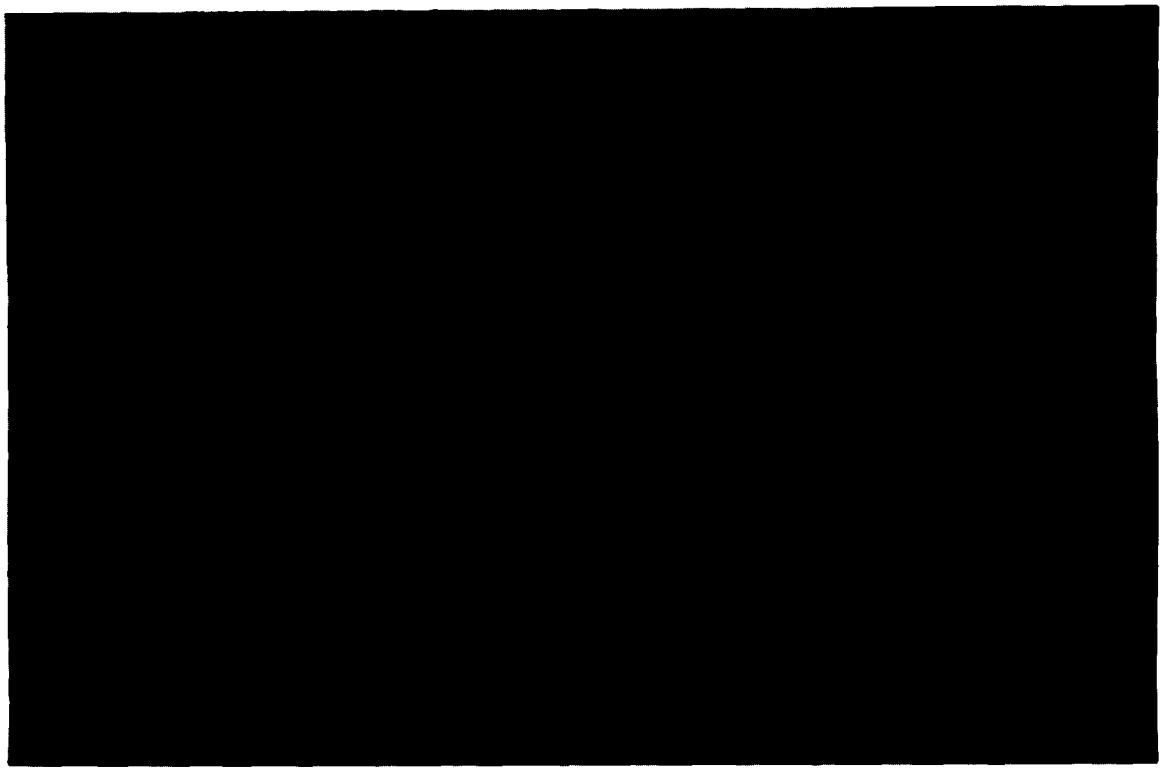

Figure 4-24: Reports using windowed Displays

Figure 4-23 is a typical bar chart produced by KBS. Figure 4-24 shows how a summary report can be displayed using multiple windows.

4.4. Experiment Rating

The **inet-pc-goal1** discussed in the example on rating scenarios was actually implemented on a model of a distribution network. After a few days of simulated time the goal was evaluated and reported.

Figure 4-25: Example of Goal Evaluation

Figure 4-25 shows a goal evaluation report produced by KBS for the CDS model.

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 0.8 which is *good*. However the overall goal rating is still unsatisfactory because of its negative rating.

```
Overall goal rating
   = wt. avg. of individual constraint ratings
   = (0.3 * 0.8  + 0.7 * -0.9)/(0.3 + 0.7) = - 0.39
```

It has been shown in detail how goals, constraints and instruments work in harmony to rate model scenarios. This means we can collect data from a model run, define and connect goals to the model and be able to tell how good the model is behaving with respect to the organizational goals. 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 the help of a Kiviat chart like the one shown in Figure 4-26.

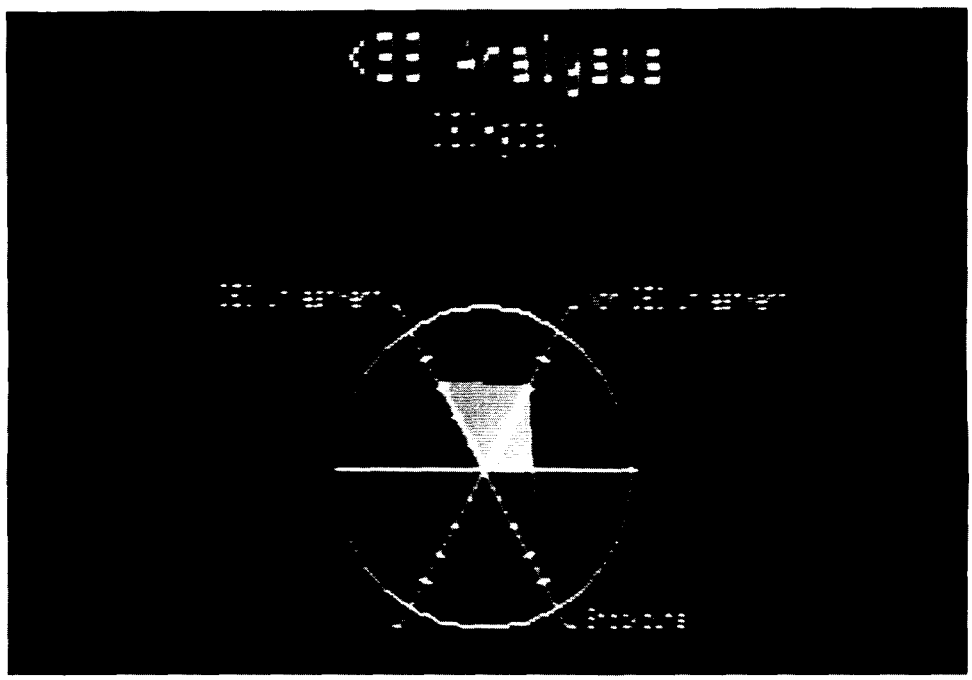

Figure 4-26: Complex Goals viewed Graphically

In Kiviat graphs all performance parameters that are "good" when they assume large values are plotted above the X axis while the performance parameters which are "bad" when they are large are plotted below the X axis. The shape derived by connecting these parameters can quickly present a view that shows whether the current scenario is good or bad (bad scenarios have large areas below the X axis).

5. Automatic Analysis of Data

A major goal of our research in KBS has been to use knowledge to automate the analysis of data generated by simulation experiments and to suggest ways in which the model can be altered to further optimize the goals and constraints. One method is to use an expert systems approach where a set of experts are interviewed in order to identify and codify their expertise. We believe this is an important first step; any knowledge not widely available to the simulation community at large, which can be made available as part of the simulation system, can have an important impact. An example of this approach can be found in Simulation Craft [Fox et al. 86] where expertise is used to identify and correct bottleneck situations. In figure 5-1, the report window (upper left) defines the goals to be satisfied by the experiment (e.g., cost, machine breakdown, resource utilization), the exception window (lower right) identifies the constraint deviations (e.g., poor utilization of machines), and the suggestions window (lower left) recommends a change to the model to maximize the goals (add more machines).

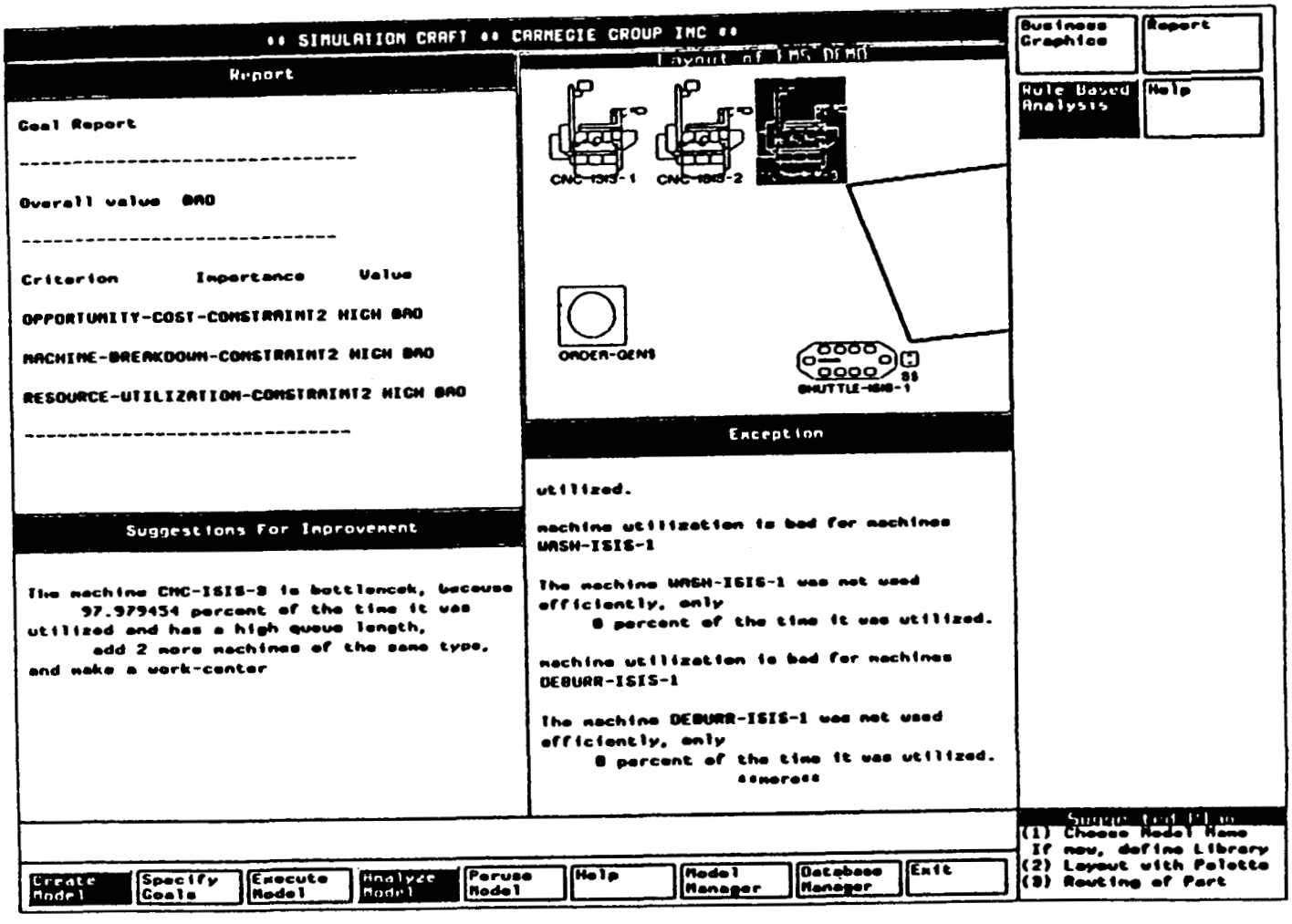

Figure 5-1: Simulation Craft Analysis and Treatment

In KBS, we have extended this approach. Recognizing that many of the systems which the user wishes to model are quite complex and possibly unique, the amount of available expertise may be limited. Though general rules which describe the causal relations among variables may exist, they may be too general to capture the details of the particular system being modeled. Our approach takes rule based expertise as a starting point. By analyzing data gathered by event tracking using a procedure known as path analysis, we are able to learn refinements for the rule set.

5.1. Learning Rules Using Path Analysis

Path analysis was originally introduced by Sewall Wright [Wright 21] [Wright 34] [SPSS 78], as a method of analysis by which the qualitative knowledge that we have regarding causal relations may be combined with the quantitative knowledge of the degree of relationship furnished by correlation and regression. In other words it is a method of measuring the direct influence along each separate causal path in a causal network of variables in a system, and thus of finding the degree to which variation of a given effect is determined by each particular cause. It must be emphasized that the method of path coefficients is not intended to accomplish the impossible task of deducing causal relations from the values of the correlation coefficients. However, in cases in which the causal relations are uncertain the method can be used to find the logical consequences of any particular hypothesis.

In KBS, path analysis is used to refine the heuristic knowledge provided by an expert. In particular, it is used to elaborate the simple causal relations normally found in rules to include interactions among variables found "buried" in the model, and to refine the degree to which variables are causally related. In other words, KBS performs a type of learning where heurstic knowledge is refined based upon the outcome of simulation experiments. The following outlines the tasks in the learning process.

- A Rule is proposed by the domain expert.
- The causal assumptions on which the rule is based 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 analysed. The most appropriate causal structure is chosen, and the results summarized to yield an equation that reflects the sensitivity of the Output parameter with respect to the controllable Input parameters. Thus path analysis is used to find the degree to which variation of a given effect is determined by each particular cause in the system being modeled.
- The sensitivity information is used to quantitatively refine the rules.

5.2. A Detailed Example

In the CDS domain several areas for analysis were isolated, namely Inventory Policy, Capacity Planning, Topology Planning, and Operational Planning. Of these Inventory Planning, has been singled out for Path Analysis mainly because the topic is familiar and numbers are easier to relate to. A typical example will best illustrate the series of steps that are taken to arrive at a refined rule starting from a more general rule proposed by the domain expert.

Example of a Diagnosis/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 and we proceed with the inet-pc model of the CDS domain shown in Figure 2-1.

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 upon the CDS model *D1:stockouts* (% of orders not filled) is an Output parameter and *M1:production-rate* is an Input parameter. A few causal structures, as shown in Figure 5-2, are then proposed which attempt to include the variables in the rule causally connected to each other along different paths.

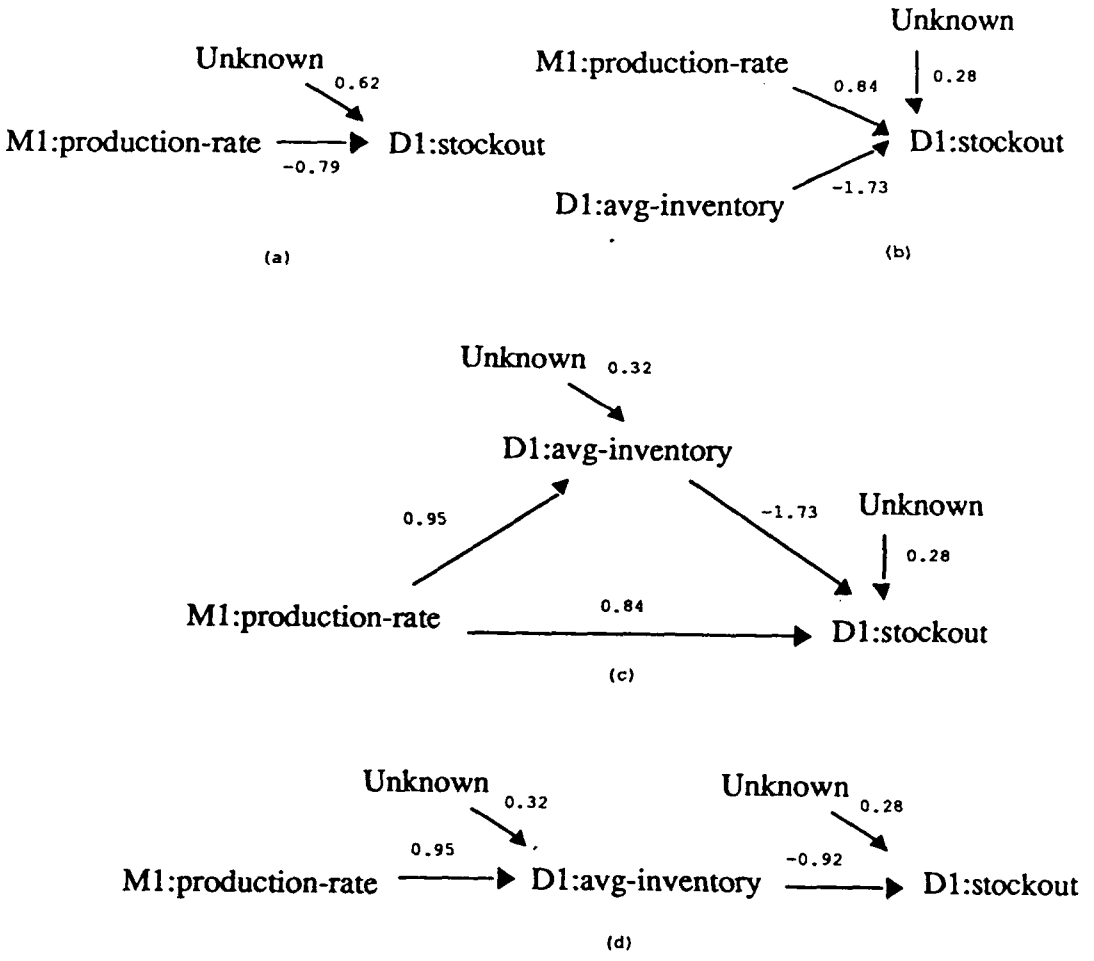

Figure 5-2: Causal Hypotheses in inet-pc model

We first verify with the help of event tracking whether *D1:Avg-Inventory*, *D1:stockouts* and *M1:production-rate* are indeed causally connected to each other. For example in Figure 5-2 (b) we must verify whether *M1:production-rate* is connected via an events chain to *D1:stockouts* and *D1:avg-inventory*, and similarly if *D1:avg-inventory* is connected to *D1:stockouts*. The causal chain connecting *M1:production-rate* to *D1:stockouts* is shown in Figure 5-3.

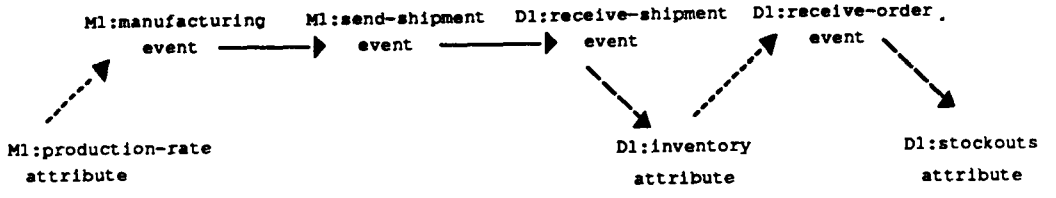

Figure 5-3: Causal Chain derived from Event Tracking

As can be seen from Figure 5-4 the causal assumptions for all Figures 5-2 (a) (b) (c) and (d), are valid.

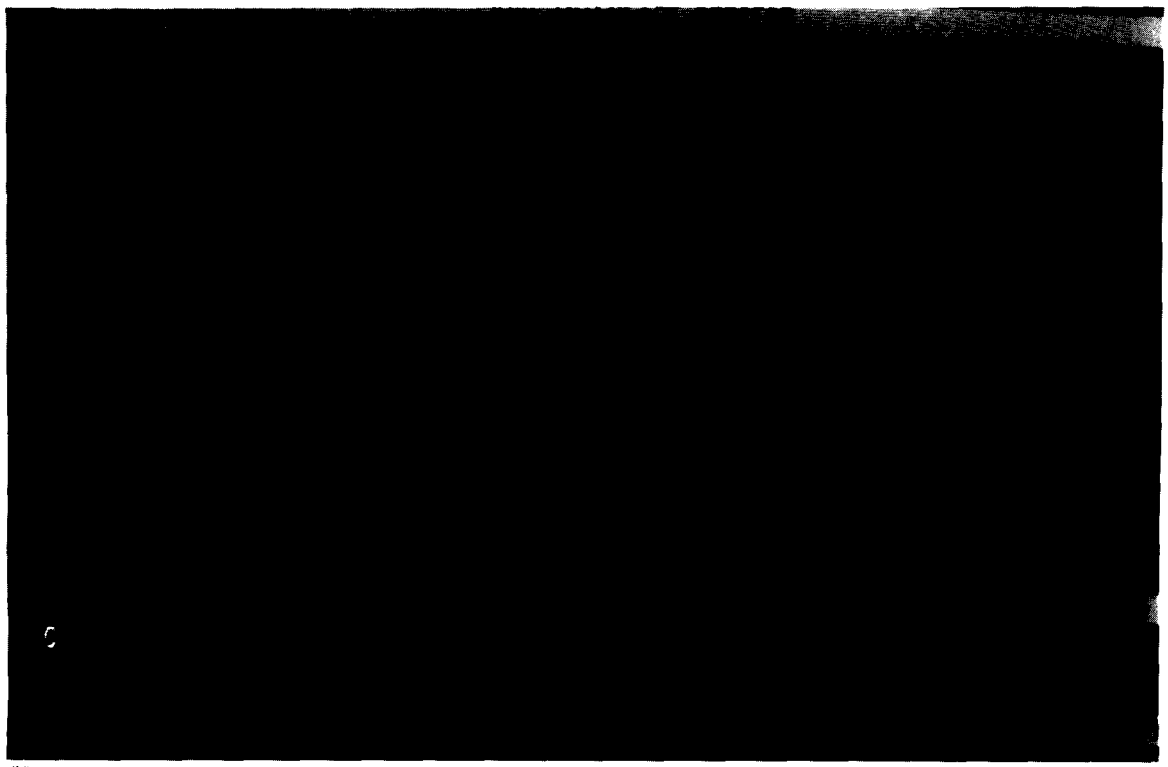

Figure 5-4: Validating hypotheses

Figure 5-2 (a) is purposely included to show the naive approach which would be taken in the absence of Path Analysis, by ignoring the intermediate variable *D1:avg-inventory*. It must be stressed here that the search algorithm used to discover causal paths can find all paths existing between two model variables given time and resources but currently it returns with the first path found. Thus the causal chain in Figure 5-3 is not the only possible path between *M1:production-rate* and *D1:stockouts* and it would be wrong to place great faith in that path and recommend that the correct causal structure from the several alternatives in Figure 5-2 is (d).[6]

A series of experiments are then designed to measure *D1:avg-inventory* and *D1:stockouts* while changing *M1:production-rate* and results of these experiments are shown in Table 5-5.

[6]It is a coincidence that in this example it turns out that it is the most appropriate hypothesis.

Experiment	*M1:Production Rate*	*D1:Stockouts*	*D1:Avg-Inventory*
1	30	48.592	11.41
2	50	48.449	14.24
3	60	48.470	13.66
4	70	48.290	14.47
5	80	48.205	17.30
6	100	48.122	15.90
7	120	48.248	17.99
8	140	48.152	19.54
9	160	44.256	25.09
10	180	41.264	28.68

Figure 5-5: Results of Experiments on inet-pc model

Using the above results, all alternate causal structures (hypotheses) Figure 5-2 (a) through (d) are subjected to Path Analysis and the coefficients in the Path Diagrams are computed. Example schemata from the causal structure representing hypothesis **h1** are shown in Figures 5-6, 5-7, 5-8 and 5-9.

{{ **h1**
INSTANCE: causal-network
MODEL-NAME: inet-pc
comment: model for which causal analysis is being conducted
NODES: stockouts1 inventory1 production-rate1
comment: a set of model variables to be studied
PATHS: inventory1-stockouts1 production-rate1-to-stockouts1 production-rate1-to-inventory1
comment: a set of computed causal paths in the network
ANALYZE-PATHS: do-path-analysis
comment: a method for conducting path analysis }}

Figure 5-6: An Example Causal Hypothesis

{{ **stockouts1**
INSTANCE: causal-node
NODE-OF: h1
comment: The causal network this node belongs to
CAUSES: nil
comment: The nodes caused by this node
CAUSED-BY: inventory1 production-rate1
comment: The nodes which cause this node
DATA: stockout
comment: The data schema which contains observations and information for mapping data in the model to data suitable for causal analysis
PATH-EQN: 39.629 - 0.837 production-rate1
comment: A symbolic equation relating this node to all the *Input* nodes on which it directly or indirectly depends on. An *Input* node is a model variable which can be altered
REGR-EQN: 48.62 - 0.45 inventory1 + 0.0167 production-rate1
comment: A symbolic equation relating this node to immediately preceding nodes on which it directly depends on.
RESIDUE: 0.116
comment: the *unknown* component expressed as a standardized path coefficient }}

Figure 5-7: An Example Causal node

{{ **Inventory1-to-stockouts1**
INSTANCE: causal-path
FROM-NODE: inventory1
comment: the causing node in the path
TO-NODE: stockouts1
comment: the effect node in the path
PATH-COEFFICIENT: -1.152
comment: the standardized path co-efficient }}

Figure 5-8: An Example Causal Path

{{ **stockout**
INSTANCE: model-variable
OBSERVATIONS: 36.37 31.95 22.87 25.44
comment: a list of values of the translated model variable, one for each experiment
ACTUAL-VARIABLES: (D1 total-orders) (D1 back-orders)
TRANSLATE-RESULTS: extract-stockouts
comment: a message to fill up the observations slot from the results of experiments
MODEL-NAME: inet-pc
comment: name of model from where data is to be extracted }}

Figure 5-9: The Model variable schema

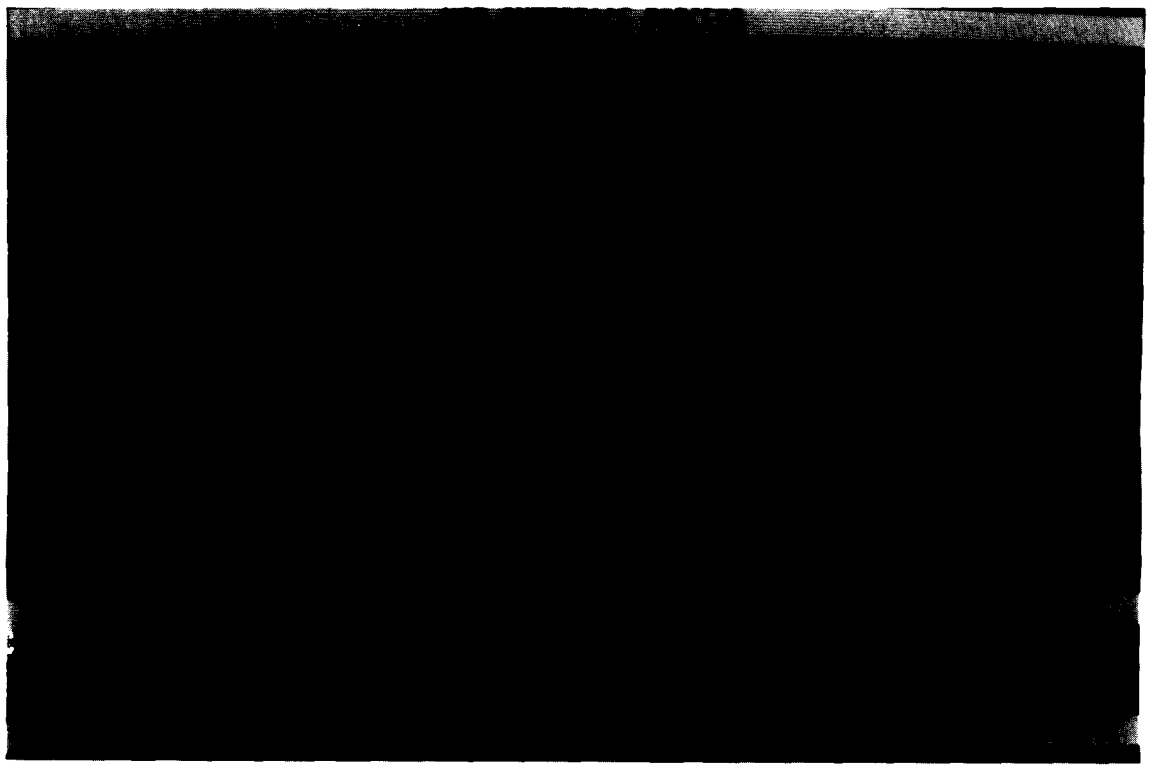

Figure 5-10: Analyzing Hypotheses

Figure 5-10 shows the regression equations derived from various hypotheses (the schemata for h1 are shown in figures 5-6 through 5-9). The residual (unknown) influence on D1:stockouts 0.62 in (a) is higher than that in (b), (c), and (d) which is 0.28, thus suggesting that hypothesis (a) should be dropped from further consideration. From among (b), (c) and (d) we reject (b) and (c) 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. Thus the most appropriate hypothesis is Figure 5-2 (d) and from the unstandardized path equations we derive the sensitivity information necessary to improve the rule.

The Regression equations:

D1:stockouts = -0.418 *D1:avg-inventory* + 54.65
D1:avg-inventory = 0.102 *M1:production-rate* + 7.693

Summarized Path analysis:

D1:stockouts = - 0.0428 *M1:production-rate* + 51.44

Sensitivity :

δ(*D1:stockouts*) /δ(*M1:production-rate*) = - 0.0428
δ(*M1:production-rate*) /δ(*D1:stockouts*) = - 23.382

Without Path Analysis we would have assumed:

D1:stockouts = - 0.038 *M1:production-rate* + 51

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* + 23.38 (*D1:stockouts* - 40)

As seen above in simple cases Causal Path Analysis reduces to ordinary Linear Regression, and by plugging in the value of the Input (causing) variable into the regressed equation we can predict the expected value of the Output (caused) variable. However in a more complicated case such as that of Figure 5-2 (d) a Causal Correction is applied to the Regression coefficients before predictions can be made. For a detailed treatment on the subject of Path Analysis refer to [Wright 21] [Wright 34] [Li 75] [SPSS 78].

6. Automating the Simulation Life Cycle

A important component of a knowledge based simulation system is the reasoning architecture used to manage the simulation life cycle. Both KBS and Simulation Craft [Fox et al. 86] use a goal directed reasoning architecture where each stage of the simulation life cycle is defined as a separate goal with a set of rules which:

- identify sub-tasks to be performed
- initialize each sub-task at the appropriate time
- identify when the sub-task is completed
- signal when the entire goal is completed

In figure 3-3, the bottom right window lists the current sequence of sub-tasks being managed by the system. In addition, unless required sub-tasks are completed in one stage, the user cannot move on to the next. For example, in figure 3-3, the upper right window defines a set of "pending activities" remaining to be completed before the model is complete enough to be simulated.

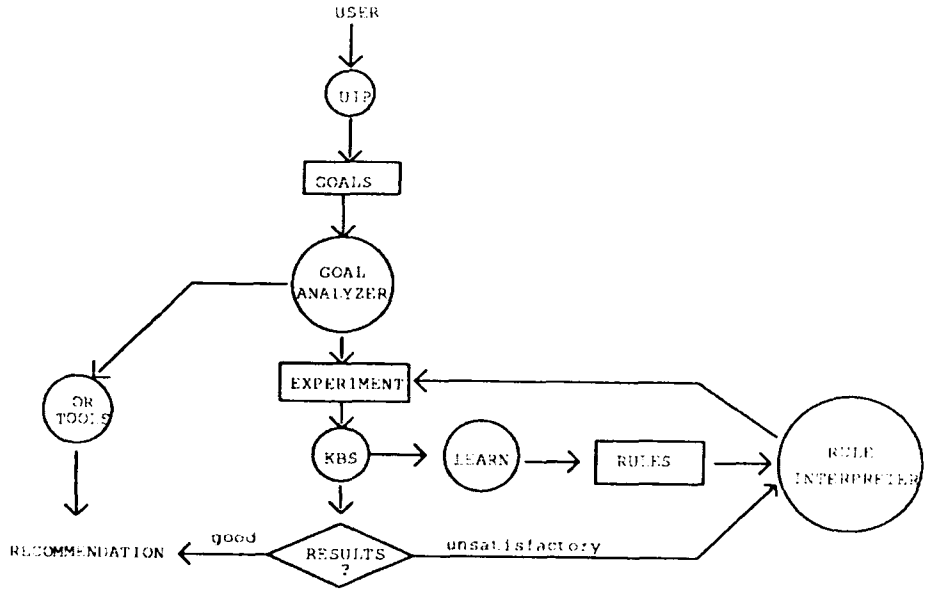

Figure 6-1: Architecture of a KBS based Expert System

Figure defines the goal network in KBS. The following outlines the sequence of activities performed by the user.

- The user interface process (UIP) 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* (i.e. a goal oriented request), the request is analyzed by a Rule-based **goal-analyzer**.
- This analysis may result in invoking an appropriate Operations Research tool, or 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" which is used in analysis.
- After an experiment is conducted the results are analyzed. If they are satisfactory with respect to the goal, i.e. a high rating was achieved for that scenario, then a recommendation is made based on the best scenario. If the results are not satisfactory then the analysis/correction rules are fired, which may generate change specs for the next experiment to be conducted.

Several types of analyses can be performed by KBS, in order to satisfy a specified goal:

- **Static Analysis.** This is used when no time dependent information plays a role in the analysis.
- **Performance Analysis.** This is used when one of the following is requested:
  - **Rating of a given scenario.**
  - **Analysis.** This is used to detect the causes of unsatisfactory behavior of the model.
  - **Scenario Generation.** This is used when a number of scenarios have to be tried before a prescription can be provided.

- **Trade-off.** This is used when it is necessary to compare several given 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 simulation, an appropriate analysis tool will be invoked.

Figure 6-2 shows a log of the execution of a simulation model which was automatically analyzed resulting in the construction of the improved scenario **kbs-experiment-2**.

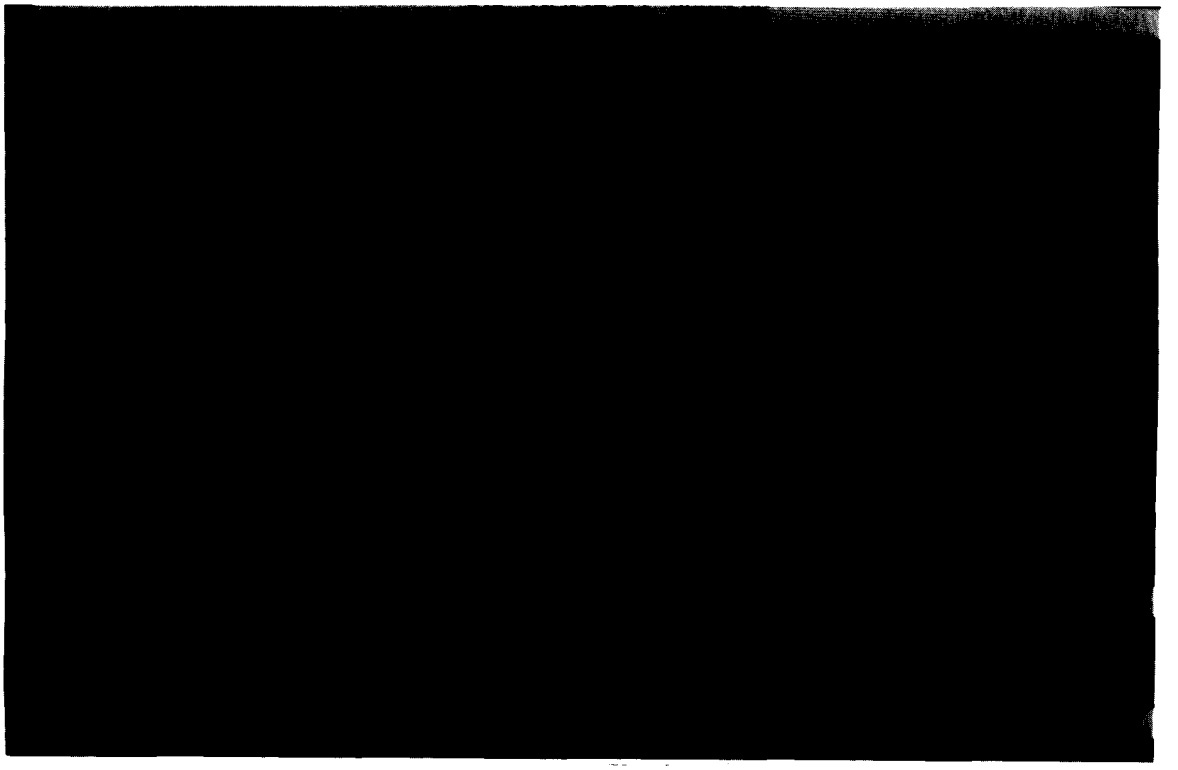

Figure 6-2: Recommendation from Automatic Analysis

The run specifications that produced this analysis are shown being entered by the user in picture 6-3.

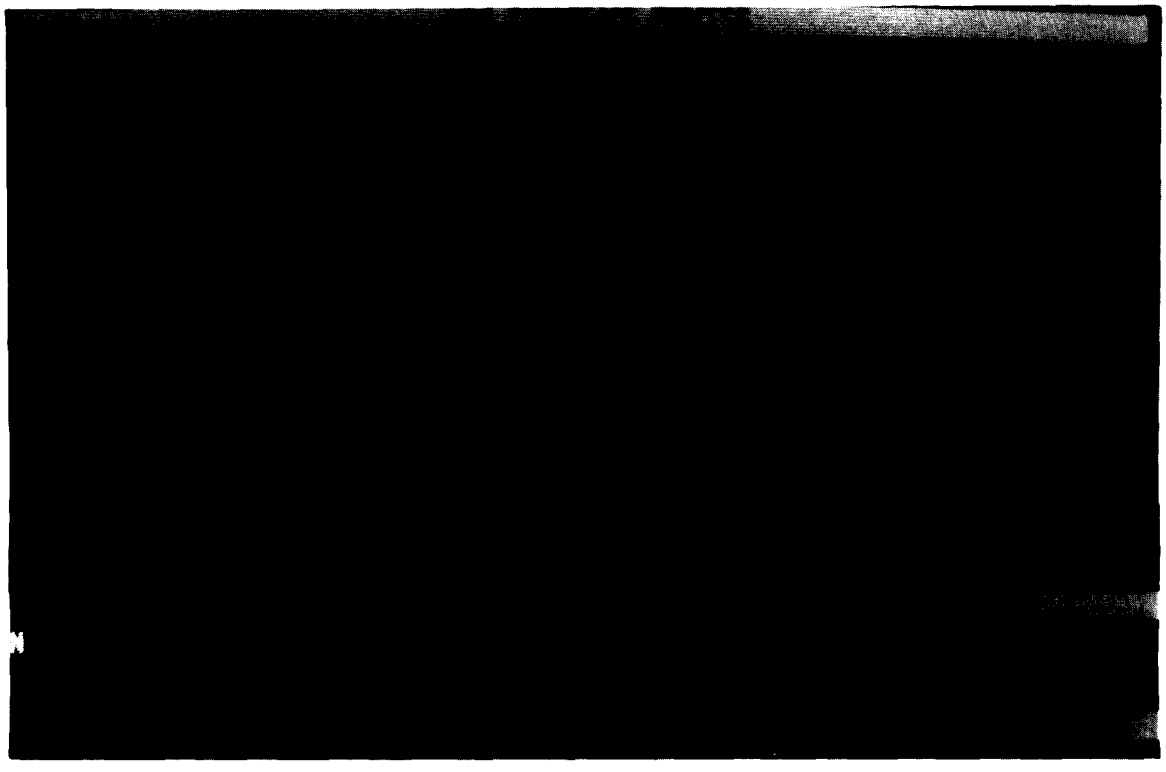

Figure 6-3: Simulation Run Profile

7. Conclusions

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 them. In retrospect, the first step of using a schema representation and a combined object and rule-based programming paradigm for representing simulation knowledge and behavior was perhaps the simplest and most intuitive. It made model creation more easy because models could be built using objects specific to the domain. It also made understanding the model easier. But there is a limit to both the ease of creation and understanding. As models became more complex the use of simple schema editors did not suffice 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. First, KBS provides a goal directed architecture which manages the life cycle enabling the clean interleaving of the reasoning performed by the user and the embedded expertise. Secondly, KBS provides an automated analysis capability where not only rule based expertise can identify and recommend

corrections for problems, but rule refinements can be learned by analyzing simulation data. The purpose of performing simulation is to produce data, which when analyzed will identify interesting aspects of the model. It is often the case that the data is voluminous and the analyst 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 time in the simulation life cycle, and to increase the quality of the results by making available expertise not readily available to the end user. KBS represents a good first step along this path.

8. Acknowledgements

The authors thank Digital Equipment Corporation for supporting this and other related research projects in the Intelligent Systems Laboratory. Thanks are also due to Dr. Donald Butcher and Dr. Stanley Wearden for suggesting the use of the Path Analysis approach to Causal Analysis.

References

[Fox 79] M.S. Fox,
On Inheritance in Knowledge Representation.
In *Proceedings of the International Joint Conference on Artificial Intelligence*. 95 First St., Los Altos, CA94022, 1979.

[Fox 83] M.S. Fox.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
Technical Report, Carnegie-Mellon University, 1983.
CMU-RI-TR-85-7,Intelligent Systems Laboratory,The Robotics Institute, Pittsburgh,PA.

[Fox 86] M. S. Fox.
Observations on the Role of Constraints in Problem Solving.
In *Annual Conference of the Canadian Society for the Computational Studies of Intelligence*. University of Quebec Press, 1986.

[Fox & Reddy 82] Fox, M.S., and Reddy, Y.V.
Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation.
In *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation*, pages 675-683. University of Pittsburgh School of Engineering, April, 1982.

[Fox et al. 86] M. S. Fox, N. Sathi, V. Baskaran, J. Bouer.
Simulation Craft: An Expert System for Discrete Event Simulation.
In *Simulators III: Proceedings of the SCS Simulators Conference*. The Society for Computer Simulation, San Diego, CA, 1986.

[Kahn 87] Kahn, G.S.
From Application Shell to Knowledge Acquisition System.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 355-359. Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, CA 94022, August, 1987.

[Klahr & Fought 80]
P. Klahr and W.S. Fought.
Knowledge-based Simulation.
In *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence*, pages 181-183. Stanford CA, 1980.

[Knowledge Craft 85]
Knowledge Craft
Carnegie Group Inc, 650 Commerce Court, Station Square, Pittsburgh PA 15219, 1985.

[Li 75] C.C.Li.
Path Analysis - a primer.
Boxwood Press, 183 Ocean View Blvd, Pacific Grove, CA 93950, 1975.

[McArthur & Sowizral 81]
D. McArthur and H. Sowizral.
An Object Oriented Language for Constructing Simulations.
In *Proceedings of the International Joint Conference on Artificial Intelligence*. 95 First St., Los Altos, CA94022, 1981.

[McRoberts, Fox & Husain 85]
McRoberts, M., Fox, M.S., and Husain, N.
Generating Model Abstraction Scenarios in KBS.
In *Artificial Intelligence, Graphics, and Simulation: Proceedings of the 1985 SCS Multiconference*, pages 29-33. The Society for Computer Simulation, San Diego, CA, 1985.

[Reddy & Fox 82a] Reddy, Y.V., and Fox, M.S.
KBS: An Artificial Intelligence Approach to Flexible Simulation.
Technical Report, Carnegie-Mellon University, 1982.
CMU-RI-TR-82-1,The Robotics Institute,Pittsburgh, Pa.

[Reddy & Fox 82b] Reddy, Y.V., and Fox, M.S.
Knowledge Representation in Organization Modeling and Simulation: A Detailed Example.
In *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation*, pages 685-691. University of Pittsburgh School of Engineering, April, 1982.

[Reddy & Fox 83] Reddy, Y.V., and Fox, M.S.
KBS: A Knowledge Based Simulator - User's Manual.
June, 1983
CMU Robotics Institute, Internal Working Document, June 1983.

[Reddy et al. 83] Reddy, Y.V., Fox, M.S., Doyle, K., and Arnold, J.
INET: A Knowledge Based Simulation Model of a Corporate Distribution System.
In *Proceedings of the IEEE Conference on Trends and Applications*, pages 109-118. IEEE, Gaithersburg, MD, May, 1983.

[Reddy et al. 86] Reddy, Y.V., Fox, M.S., Husain, N., and McRoberts, M.
The Knowledge-Based Simulation System.
IEEE Software : 26-37, March, 1986.

[Reddy, Fox & Husain 85]
Reddy, Y.V., Fox, M.S., and Husain, N.
Automating the Analysis of Simulations in KBS.
In *SCS Multiconference on AI, Graphics and Simulation.* January, 1985.

[Rychener 82] M.D. Rychener.
PSRL: An SRL-Based Production Rule System - Reference Manual for PSRL Version 1.2.
Technical Report, Carnegie-Mellon University, 1982.
CMU-RI-TR-85-7, The Robotics Institute, Pittsburgh, PA.

[Sathi, Fox & Greenberg 85]
Sathi, A., Fox, M.S., and Greenberg, M.
Representation of Activity Knowledge for Project Management.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-7(5): 531-552, September, 1985.

[Sathi, Morton & Roth 86]
A. Sathi, T. E. Morton, and S. Roth.
Callisto: An Intelligent Project Management System.
AI Magazine : 34-52, 1986.

[SPSS 78] *SPSS Manual*
1978.

[Venkataseshan & Reddy 84]
Venkataseshan, B., Reddy, Y.V.
An introspective environment for Knowledge Based Simulation.
In *Proceedings of the Winter Simulation Conference.* 1984.

[Wright 21] Sewall Wright.
Correlation and Causation.
Journal of Agricultural Research , 1921.

[Wright 34] Sewall Wright.
The Method of Path Coefficients.
Annals of Mathematical Statistics , 1934.

[Wright & Fox 83] J.M. Wright, and M.S. Fox.
SRL/1.5 User Manual.
Technical Report, Carnegie-Mellon University, 1983.
Intelligent Systems Laboratory,The Robotics Institute, Pittsburgh,PA.