Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

MODELS AND ARTIFICIAL INTELLIGENCE

by

Robert E. Shannon, Ph.D, P.E.

Industrial Engineering Department
Texas A &M University
College Station, Texas, 77843, U.S.A.

ABSTRACT:

The Art and Science of modeling and simulation is rapidly changing. Recently there has been an increasing interest in the possibilities of incorporating the technologies developed by the artificial intelligence research community into the modeling and simulation process. This paper addresses: (a) the nature of artificial intelligence, (b) it's present and potential applications in the design and management of modeling systems, (c) the potential benefits of this technology over existing approaches, and (d) the current state-of-the-art as it applies to simulation.

INTRODUCTION:

Ever since the advent of the modern digital computer, scientists have speculated about and argued over the possibility that computers could be made to behave in a way that would be perceived as intelligent. Artificial Intelligence (AI) has its roots in the speculative essays by Turing on the powers of computers. AI as it is known today, is the result of a meeting convened in 1956 by ten scientists interested in symbolic computation at Dartmouth College. The Dartmouth conference represented the beginning of AI as a separate and distinct aspect of computer science. Thus it is important to recognize that the fields of artificial intelligence (AI) and expert systems (ES) is over 30 years old.

The purpose of this paper is to discuss the potential of modeling and simulation environments based upon Artificial Intelligence and Expert Systems technology. Such systems will hopefully allow models to be quickly developed, validated and run with as much of the necessary expertise as possible built into the software. Expert simulation systems [Shannon et al 1985, Adelsberger et al 1986, Arons 1983, Gaines & Shaw 1985] and the application of logic programming offer the potential for a new generation of substantially more powerful methods for modeling and simulation. Such AI/ES based modeling systems will follow a different paradigm than that currently used. The goal is that in the future, the modeler will declare the knowledge about the system , define the goal and let the computer work to find the solution by defining the experiment to be run, the correct model, execution of the experiment and reporting the results.

NATURE OF ARTIFICIAL INTELLIGENCE:

Before getting into a discussion of AI applications in modeling and simulation, we should first define AI. McCarthy (who coined the term at the 1956 meeting mentioned above) characterized artificial intelligence as the part of computer science concerned with designing intelligent computer systems that for certain limited areas, emulate some of the characteristics associated with intelligent human thought. Among these are the ability to learn, reason, solve problems and understand ordinary human language.

It is important to separate the discipline or science of AI from the application of the results of this research. The discipline of AI has since its inception been concerned with trying to understand how humans beings acquire, organize, store and use knowledge. In order to test and prove it's theories, the artificial intelligence research community has formulated computer implementations of their models. These implementations take the form of specific algorithms, data structures, and programming languages which try to emulate the knowledge which a human uses and the cognitive processes with which the human manipulates this knowledge. The applications aspect of artificial intelligence focuses on attempting to apply these theories and their resulting computer implementations to solve practical problems in a real world environment. The applications of AI research thus fall into two general classes:

(1) Duplication of natural human capabilities.

(2) Duplication of learned skills and expertise.

The first class of AI applications, the duplication of human abilities, includes such areas as language processing, vision, reasoning, sensory fusion, scene analysis, touch sensing etc.. Example applications of language processing include command interpretation, speech recognition, natural language database query, and automated documentation generation. These and the other areas of the first class of applications have broad applicability, particularly in the areas of process automation and robotics.

The second class of applications is concerned with attempting to duplicate learned skills and expertise. This class of applications is usually

referred to as "expert systems." These applications are concerned with the automation of tasks that are normally performed by specially trained, talented and experienced people in some specialized problem domain. In other words, expert systems try to provide one person with interactive access to another person's skills and knowledge, encoded in a computer. The primary goal of Expert Systems (ES) is not the understanding of the basic mechanisms used by the human expert to arrive at a given result, but rather, it is to consistently duplicate the decisions which would be reached by the human expert. Even though the first class has broad applicability, the remainder of this paper will concentrate on "expert systems", as they offer the most direct opportunities for improved modeling and simulation support.

DIFFERENCES:

A very real question for us to consider is what is the reality of AI/ES technology? Is it all smoke and mirrors or is there something new and useful? An immediate problem for anyone first getting interested in AI/ES is the terminology. "Knowledge Engineering" means modeling, "Knowledge Base" means database, "Backward Chaining" is the concept of dynamic programming using symbols as well as numbers, "Production Rules" are IF-THEN or IF-THEN-ELSE constructs, "automatic programmers" are compilers, demons are conditional events etc. When the truth is told, expert systems techniques turn out to be mostly the same as operations research techniques, and good software engineering put together in some clever ways.

Once you get beyond the buzzwords and hype, there's less difference than you might think between conventional computer programs and so-called expert systems. Because artificial intelligence and expert systems have become so much in vogue, it has become fashionable to refer to any computer program which uses LISP, PROLOG, object oriented programming, rules or any of the myriad programming languages/shells as being "knowledge-based" or "AI based." All software embeds knowledge and certainly a simulation model written in any of the existing modeling systems represents the knowledge or understanding of the modeler about the system being studied. The so-called "expert system tools" or "shells" that are so widely discussed in the technical press are really nothing more than 1) a programming language for encoding and applying knowledge, and 2) a special environment that facilitates program development.

Most of AI's successes and contributions have historically been from it's spin-offs; fancy user interfaces, timesharing, spelling checkers, screen editors, pointing devices and so on. All of these have come about as a result of AI researchers trying to create a friendlier environment in which to work. The ultimate goal of any programming paradigm is to close the gap between what users conceptualize as a representation of a system to be simulated and how they actually express that relationship in some sort of computer executable form. Despite the proliferation of terms, almost all AI/ES simulation systems under development today utilize object oriented programming, rules and graphics. We will therefor first discuss these concepts and then discuss some representative systems currently under development.

OBJECT-ORIENTED PROGRAMMING:

The basic unit of information and organization in most AI/ES based simulation systems is the "object." Object oriented programming originated not from AI research but rather from the simulation language SIMULA [Dahl and Nygaard 1966]. Today it plays a key role in most AI/ES systems. Although it has assumed a number of different names in the literature (e.g. concepts [Lenat 1976], frames [Minsky 1975], schemas [Bartlett 1932], actors [Klahr et al 1980], flavors [Allen et al 1983] and units [Bobrow & Winograd 1977]), the underlying notion of the object is to organize and store pieces of information relating to a single concept into a single location. The pieces of information may include facts about the object, how the object behaves under certain stimuli, and with whom the object interacts.

Objects are structurally organized as a collection of slots. Each slot corresponds to a piece of information related to the object. Slotnames identify each slot and are public to all objects. The information associated with each slot, however, is internal to the object itself and is hidden from the other objects. It is obtained by sending a message requesting the information by slotname.

Object-oriented programming provides a way to elevate model and experiment representation to a higher level of abstraction and a more natural form of representation than is possible with today's procedural oriented simulation languages. Object-oriented programs are written in terms of "objects" (also referred to as schemata or frames) rather than in terms of procedures. In this style of programming, knowledge about the objects (facts as well as how the object is to do things) is associated with the objects themselves. The philosophy of object-oriented programming is a simple one, and directly supports the simulation problem solving approach, especially for systems that deal with the explicit passage of time and/or changes of objects in time. This can be summarized as follows:

(1) The user first creates or defines objects that correspond to real world objects, and represent modular components of the real world.

(2) The behavior of the simulation model's objects describe the behavior of the real world objects and how these objects will behave/perform in response to various inputs.

(3) Objects act on each other by passing messages describing both functional and relational actions. Messages passed between objects are carriers for all interaction between objects.

Thus, object-oriented programming treats a program as a collection of objects that perform actions by sending and receiving messages. In essence, the object oriented "world" of this simulation environment consists of packets of information that provide behavioral rules (object embedded) and manipulation specifications (message embedded). The object-oriented approach is especially valuable in that it provides a close correspondence between simulated objects and real world objects.

Furthermore, once an object is defined, it may serve as an abstraction for additional objects. In practice this is done by grouping objects that do the same things in the same way into "classes" (also sometimes called "flavors" or "types"). An object's class specifies the attributes of the object such as the kind of data it stores and the possible actions that can be performed on the data (known as "methods" or "behaviors"). A single composite object may be created that behaves similar to a group of more primitive objects. Thus a complex hierarchy of objects with inherited properties and behavior rivaling real world situations may be modeled.

The power of object-oriented programs comes from two features of classes: encapsulation and inheritance. Encapsulation refers to the fact that a specific type of data and the means to manipulate it can be combined in a class. Inheritance means the classes can be organized in a tree-like fashion (called an inheritance hierarchy or semantic network) so that new classes can inherit information (facts and methods) from their ancestors [Salzberg 1987]. These features also make software written in object-oriented languages highly reusable.

One of the powerful aspects of object-oriented representation is its ability to conserve information when describing objects. As we have seen, objects may be defined in terms of other objects. This is similar to the approach taken by Simula [Dahl and Nygaard 1966] and Smalltalk [Goldberg 1984]. Objects consist of slots which contain values. New objects may be defined so as to inherit slot values from previously defined objects. The new object may be specialized with the addition of new slots. A single composite object, then may be created that behaves like a group of more primitive objects.

Not only may object definitions be structured hierarchically, but object behaviors as well. This concept reflects a structured approach to model composition in which objects at one level of abstraction perform no productive behavior other than to activate objects at a lower level of abstraction. Data encapsulation is preserved since objects at the lower levels are autonomous and need not know what initiated their actions. Objects are only aware of message transfers, thus defining the scope of communication. This top down approach allows the user to readily model real systems in any level of detail. Moreover, it permits pieces of models to be simulated for verification and validation purposes.

As an example, consider a work station consisting of a drill and a lathe. The two sentences "load the drill with part xyz" and "load the lathe with part xyz" both have the same abstract meaning. That is, it is desired to perform an operation on a machine. In this case the operation is to load part xyz. However, loading a lathe is different from loading a drill and would require different times to accomplish, even though the word 'load' is the same in both sentences. In most implementations of object oriented programming, the definition of an object is of the general form:

object(<name>, <properties>, <behaviors or methods>)

A property represents a fact about the corresponding real world entity. For example, one type of fact might be about membership in a class. A class represents the familiar concept of a set. For example the class "lathe" represents a specific subset of the meta class "machine" which is used to turn parts. A "turret_lathe" is a subclass of the class "lathe". Thus we may describe a complex hierarchy of objects. Suppose we define an object:

```
object: machine
  is-a: equipment
  type: stationary
```

Machine is defined as an object with two properties one of which is being stationary. This is a property that we want to be true of all child objects of machine. It is further described as being a subclass of a more abstract object called equipment which, in turn may have properties. Now if there is another object:

```
object: drill
  is-a: machine, workshop
  capacity: 1
  load_time: 5
  status: idle
  behaviors: load, execute, unload
```

then, the drill also inherits the property of being stationary and any other properties that the objects "equipment," "machine" or "workshop" might have. Notice that an object can have more than one parent. Behaviors or methods can also be inherited In a similar manner. In this way, a complex network of characteristic and behavioral relationships can be established.

Behaviors or methods are usually in the form of rules (or algorithms) to be executed when a certain message is received. For example a behavior might be:

```
rule: load
  IF   told 'load Machine_X'
  THEN set Machine_X status to busy and
       delay for load_time and
       send message 'ready Machine_X'.
```

Objects can exhibit behavior which is dependent on either time or conditions. In the latter case, most systems distinguish between independent (active) and dependent (passive) objects [Zeigler 1976]. Active objects are monitoring the system constantly and trigger off their behavior as soon as they recognize that their conditions have been fulfilled. They are capable of influencing the actions of other objects in the model. Active objects would be such things as workers. Passive objects are objects which also wait for conditions to be satisfied but they are getting this knowledge in the form of messages sent to them. They are objects that are acted upon i.e. they require some sort of external stimuli to operate. Queues exemplify this type of object in that they retain their state indefinitely unless acted upon by another object.

An object-oriented simulation system would contain three types of objects: domain independent, domain dependent, and application specific. Domain independent objects provide behavioral definitions for a generic set of model components such as random variate generators, statistical analysis modules etc. These objects are common to all simulation models. Domain dependent objects describe model parts that correlate to real components of the system. Objects in this category, although used in a particular application, are general to the domain of

manufacturing. For example, a manufacturing simulation system would have predefined objects for workers, machines of different types, material handling systems, etc. These domain dependent objects provide the templates for the creation of specific instances of the object described. Application specific objects provide information on the specific combinations and numbers of components needed for the specific study underway as well as the sequence of model components that are activated during the execution of the model. They define transactions that are unique to a single application area or study.

RULES:

Rules (sometimes called production rules) are used for multiple purposes in a simulation environment. These are statements of the form, "IF <condition> THEN <action>". The condition is often a conjunction of predicates that test properties about the current state of the system, and the action then changes the current state. Descriptions of a given situation or context of a problem are matched to a collection of conditions in a rule that causes the rule's actions to execute, in turn giving rise to new descriptions that "produce" more actions (hence the name "production rule"), and so on until the system either reaches a solution or halts. This is not a new concept and is simply the IF-THEN construct present in most standard programming languages. A simulation system containing a set of production rules is called a production or rule-based system. The production rules are the operators in the system; they are what it uses to manipulate the database.

In a rule-based simulation system, rules can be used for at least three different purposes:

1. To define the behaviors or methods which are to be used by objects;

2. To test the model for completeness and validity;

3. To drive the model towards goal achievement.

The use of rules to define the methods objects are to use when told to do so, has already been used in the preceding discussion of object-oriented programming. The object's declaration describes a set of facts about the objects (the attributes) and a set of rules for the manipulation of those facts. The rules describe the steps that permit the assertion of new facts into the knowledge base.

Rules can also be used for model verification. Model verification can be viewed as a series of decisions about completeness of the specified model and flow of information/entities through the system. These decisions can be represented as a set of rules. An example rule might be of the following form:

```
rule: Processing_Prerequisite

IF entity A is a component of operation P AND
    P is executed at station X AND
    A is not initially located at X AND
    A is not routed to X AND
    A is not an output from an operation Q at X
THEN print an error message AND
    display all stations that have A as an
    output AND prompt for a correction to model
    specifications.
```

The third use of rules is in driving the model towards goal achievement. Assume for example that the goal is to show that the system can meet certain performance criteria. The criteria serve as goals in the sense that they describe how the model must behave to meet the user's needs: they are a measure of model acceptability. Failure to meet the performance criteria may mean that the required resources were not available. Suppose for example that one of the performance criteria (goals) was to have the utilization of all workers at between 50 and 70%. We could then specify two rules such as:

```
rule: Worker_Underutilization

    IF   Worker_Utilization is less_than .5 AND
         Number_of_Workers is > Min_Workers
    THEN reduce Number_of_Workers by one AND
         continue

rule: Worker_Overutilization

    IF   Worker_Utilization is more_than .7 AND
         Number-of_Workers is < Max_Workers
    THEN increase Number_of_Workers by one AND
         continue
```

One possible way to implement such rules would be to assign performance criteria to certain objects (e.g. workers, machines etc.) along with procedures (rules) to be executed if the object failed the criteria.

GRAPHICS IN SIMULATION:

The use of graphics has become an integral part of many simulation systems and will become increasingly important. The use of graphics is and will continue to be important to the new simulation systems. There are basically two ways in which graphics can help in simulation: (1) to facilitate model construction and debugging; (2) To display and help in the understanding of the simulation results.

There are three classes of graphics applications in simulation and modeling. One class is the use of *"iconics"* for displaying the real system on the screen. In this type of application, icons which look like the components of the system being modeled, are placed on the screen to show the physical (spatial) relationships. Such iconic models are scaled down to fit upon the screen and are often used for animating the flow of objects through the system so that the user can "see" a simulation of the system in operation. Such applications can be very useful in debugging (verification) and validation of the simulation by showing whether the results are logical and the model is behaving like the real system.

Iconic modeling can also be used for specification (definition) of the system. The user selects an icon representing the appropriate system component from a menu and places it on the screen. That action calls up a pre-defined template which the user then uses to define the action and /or logic of the component by selecting pre-defined functions or by explicit entry.

The second class of graphics application is similar to flowcharting and could be called *"logic" graphics*. Symbols are interactively placed on the screen to represent systems logic. The symbols represent modules of macro-code designed to perform a

certain computational function. This idea was first used in GPSS to aid the model developer and was later adopted by SLAM, SIMAN and others.. The picture shown on the screen does not reflect the systems components but rather represents the logical relationship between the system components. There is a one to one mapping of the symbol into a function call to a pre-written program module. The user provides the arguments to be used by the module. This type of graphics helps the user to visualize the logic flow of the model. There is very little correspondence to actual systems components. It is also possible to develop such systems where the flow of the objects thru the program are shown thru the changing of colors of the blocks. This graphical tracing can be helpful in debugging.

The third class of graphics used in simulation are those for presenting and displaying output information and simulation results, the so-called *"presentation" graphics*. Display graphics are of two types, static and dynamic. In the static mode, results of simulation runs are displayed as bar charts, line graphs, histograms, scatter diagrams, pie charts etc. Such applications are helpful in analyzing and communicating results. Several languages provide methods of presenting such results and/or of converting the output files to DIF format so that they can be used and displayed with other graphics systems such as Lotus 1-2-3 etc..

The dynamic mode of presentation graphics is also used to display an animated form of output results. Some software packages allow the user to display information such as queue lengths (by bars graphs which dynamically change their vertical length) and whether a resource is busy or idle (by color changes). In some cases this is done in a post processor package, while in others it is displayable during simulation execution. Finally, this third class of graphics can also be used for displaying such things as probability distributions, frequency diagrams etc. to help the user identify input or time delay distributions.

The use of animation as a part of simulation methodology has grown rapidly in the past few years. Although it is often thought of mainly as an aid to presentation, it is in fact beneficial in all stages of model development and use. The benefits are in three areas [Smith & Platt, 1987]:

(1) Benefits for the builder

(2) Improved communications between model builder and user

(3) Benefits in presentation to users and management

For the model builder, animation provides a powerful verification tool by speeding the process of locating and removing errors in the model. Although correct functioning of the animation is not sufficient for complete verification of the model, it is also true that many errors will signal their presence by inappropriate behavior visible in the animation. For example, if one sees two AGV's pass through each other on a single track one knows something is wrong. Many analysts have found model verification to be one of the most useful aspects of animation.

Another benefit of animation is in the increased communication it allows between the model builder and the model user during development. Written specifications are hard to understand and it is particularly hard to spot omissions, faulty logic or erroneous assumptions. It is very hard to engage the typical model user in meaningful dialog over specification documents, and technical descriptions during model development. With high resolution graphics, however, the user has a graphic depiction of his or her plant which makes sense. The logic and operations described in the specifications become animated elements whose flows and interactions can be observed and followed. The typical user becomes eagerly and actively involved in periodic reviews of the model under development and omissions or inappropriate representations can be detected and corrected early during the development.

The most obvious and recognized benefit of animation is in presentations to management. The presentation of the analysis and results is enhanced immeasurably when presented partially through animation. Animation is invaluable in communicating the nature of design flaws or problem areas uncovered during the study, as well as demonstrating the proposed solutions. This is simply a recognition of the old saying that one picture is worth a thousand words. Animation makes lively and immediate what would otherwise be a dry and sometimes obscure presentation of tables and figures.

One strong word of warning is required. The use of artificial intelligence techniques and animation have not revoked the laws of probability and statistics. Adequate sample sizes, correct experimental designs and statistically correct analysis techniques are still required to draw correct conclusions. Animation is useful for getting a "feel" for system performance but not a substitute for correct simulation methodology.

EXAMPLE SYSTEMS:

The state-of-the-art in bringing AI/ES technology to bear upon simulation is very early in the development cycle. There are basically two different approaches being pursued.

(1) Hybrid Systems - building intelligent front and back ends on existing simulation systems.

(2) New Systems - changing the simulation modeling paradigm.

A number of simulation software developers have taken the approach of developing intelligent, automatic programming interfaces to existing simulation systems. In these hybrid systems an interactive interface is developed to allow the user to describe the system to be simulated in terms of graphical icon selection/ placement, menu choices, and answering computer controlled interrogations. Such systems of necessity are limited to a specific domain such as computer networks [Murray 1986], electronic assembly [Ford & Schroer 1987], flexible manufacturing systems, [Mellichamp & Wahab 1987], AGVS's [Brazier and Shannon 1987], or manufacturing [SIMFACTORY 1987, Nyan 1987, Ulgen & Thomasma 1987 and Seliger et al 1987]. In each of these cases, the system automatically writes the model and experiment to be run in an existing simulation language such as SIMAN, Simscript II.5, Simula, Smalltalk-80 or GPSS.

The user need not be familiar with these languages and may not even be aware of what language is being used to write the model.

Beginning as early as 1974, [Heidorn 1974, 1976] and continuing to the present [Ford & Schroer 1987], there have been attempts to develop automatic programming systems in which the user enters a natural language description of the system to be modeled, the system analyzes the input and requests any additional information needed, either for clarification or completeness, and then writes the necessary computer code in an existing commercial simulation language. However because of the limited state-of-the-art in natural language understanding, these systems must severely constrain the domain and form of discourse.

Attempts are also being made to develop intelligent backends to existing simulation systems that will aide the user in analyzing the results and suggesting modifications [Nyan 1987, Seliger et al 1981 and Wadhwa et al 1987]. In these systems, a goal is set, the model executed and if the desired results are not achieved, the sympthoms are analyzed by the software and suggested modifications presented to the user. This is usually accomplished by a program consisting of a set of rules of the form (IF this symptom or condition exists THEN suggest this action). This approach has also been used to develop Expert Diagnosis programs for debugging simulation models [Hill and Roberts 1987].

The advantage of hybrid systems is that they are fairly easy to develop, and the finished model executes at a fairly rapid speed. The disadvantages are that they still follow the current paradigm i.e. they have reduced the programming task, but the user must still decide upon the scenarios to be run, interpret the results, decide upon what modifications to the model must be made etc..

Several attempts have been made to take a different approach and follow a different paradigm. The RAND Corporation developed the Rule-Oriented Simulation System (ROSS) in the late 1970's [Klahr et al. 1980]. Subsequent to it's introduction as a war-gaming development tool, it gained popularity as a general purpose simulation language in the American Department of Defense Community. ROSS is a LISP implemented, interactive system. Object oriented programming serves as a basis for ROSS which was developed specifically for war game simulations and military air battles. Real world systems are modeled as objects. Messages are passed between objects describing actions that are to be taken, If-then rules describe behaviors each object may assume. ROSS aids the user during model execution by displaying a trace of all messages passed during the simulation.

Through selective filtering of trace data, users can determine if the model is behaving appropriately [McArthur & Klahr 1982]. . The user may at any time stop the simulation, modify the model, and continue the simulation. One of the primary objectives of ROSS was to incorporate the ability to reason about the behavior of models [McArthur & Sowizral 1981], however, no publications have been found to indicate that steps have been taken to implement this capability.

Knowledge Based Simulation (KBS) is a LISP based discrete simulation system developed at Carnegie-Mellon University [Fox & Reddy 1982, Reddy & Fox 1982]. Outwardly similar to ROSS, it incorporates an object-oriented paradigm to describe the real world system to be modeled. Rules are used to describe the behavior of each object. Unlike ROSS, KBS uses a sophisticated knowledge representation scheme. In KBS, models are constructed using SRL, a frame-based knowledge representation language. All entities in KBS are represented as SRL schemata which incorporate inheritance relations. Goals describing the performance criteria of model components may be attached to objects and KBS informs the user whether goals were met. KBS is designed to be used interactively, enabling the user to examine the model and its behavior. This includes model creation and alteration, run monitoring and control, as well as graphics display. It also allows the user to define and simulate a system at different levels of abstraction, and to check the completeness and consistency of the model. Research has been conducted on ways to automatically analyze the model [Reddy et. al. 1985]. Commercial adaptations of this system are being marketed by the Carnegie Group under the name of SIMULATION CRAFT, by IntelliCorp as SIMKIT, and by the IntelliSys Corp. under the name of LASER/SIM.

A research group at the Hungarian Institute for Coordination of Computer Techniques in Budapest, developed a simulation system called T-Prolog, written in M-PROLOG [Futo & Szeredi 1982]. This system is based on an underlying theory of simulation that is quite different from the previously mentioned systems. They have combined the time handling primitives of simulation and the symbolic processing of artificial intelligence into a PROLOG superset. The resulting system is intended to allow the user to specify the model in first order predicate statements and execute the model with the non-deterministic problem solving methods of prolog. T-Prolog allows the user to specify multiple model parameters and goals the model is to achieve. The run time interpreter executes the model and attempts to find the first parameter set that meets the goals. Further refinements called TS-Prolog], have resulted in an object-oriented approach analogous to behaviors in ROSS and KBS. TS-PROLOG incorporates facilities similar to those found in conventional simulation languages. Predicates are defined which start or stop processes and to provide communication between them. Every process is formulated as a goal. One of the attractive features of this system is the use of backtracking to automatically modify the model until the simulation exhibits some desired behavior. Both continuous and discrete modeling can be handled [Futo 1984].

Researchers at the Vienna University of Economics and Administration [Adelsberger & Neumann 1985] have developed a simulation system called V-GOSS (Vienna Goal Oriented Simulation System) which is implemented in several dialects including, Waterloo PROLOG, York Prolog and micro-PROLOG. The system is a quasi concurrent PROLOG interpreter along the lines of the Hungarian approach. It is a process oriented, discrete event simulation system. The user defines the initial structure of the model and declares the goals which have to be achieved. An interpreter implements a backtracking co-routine concept.

Another logic programming language for simulation is being developed at the University of Calgary [Cleary et. al 1984, 1985]. This language is based

upon Concurrent Prolog with the addition of time delay expressions. Limited backtracking is supported in the initial implementation. This enables alternate paths in a simulation to be explored for acceptable solutions. The system allows processes to receive information from unspecified processes as well as to send messages to arbitrary processes via read only reference variables. Processes are dynamically created whenever a new goal is invoked.

Finally, an interdisciplinary team of faculty from the Industrial Engineering and Computer Science Departments at Texas A&M University has been working to design a new simulation environment [Adelsberger et al 1986, Humphress 1987, Ketchem et al 1985]. The goal of this project is to "humanize" the simulation environment and process while integrating the functionality, ease of use, ease of model creation, dynamic run time interaction, and model extensibility. The approach that the simulation group at Texas A&M University has taken is to try to combine the best of the environments. It integrates the object-oriented programming and rule-based techniques of ROSS, the knowledge base approach of KBS and the goal proving mechanisms of TS-Prolog.

The system under development is a rule based, object-oriented simulation environment having the following system design characteristics:

* Object creation performed via graphical input, template/menu input, natural language dialogue with an Intelligent Assistant, and finally, if desired by using a specification language. In any case, no programming is required.
* The simulation model as well as any simulation experiments are treated by the system just as objects.
* All interaction with the system is interactive.
* The run time environment provides for model (object) modification, run time displays, automatic experimental designs, and statistical displays.
* The model, experiment and any modifications to the objects are checked by conflict resolution for consistency and completeness.
* Goal directed simulation is used to augment the experimenter and to provide automatic modification to the simulation model.
* Graphic displays during model creation/modifications as well as of the simulation dynamics.
* Selection of various abstraction levels of the simulation model and/or experiment.

SUMMARY:

It is clear that the transition to knowledge based modeling systems is already underway [Shannon 1986]. The increasing use of interactive graphical model construction and data input; graphical and animated output analysis; the separation of modeling, experimental and output analysis frames; the embedding of more and more of the statistical analysis within the language; all of these are the first tentative steps.

Knowledge based simulation systems based on the application of artificial intelligence concepts and logic programming will hopefully generate a new, more powerful environment for simulation modeling. The goal is to simplify and put at the fingertips of the semi-naive user the expertise of the most knowledgeable and experienced simulation experts. Although there are similarities between what is being done today and the goals of an expert simulation system, there are important differences. The primary one is the desire to build into the modeling system most of the decisions that are now made by the simulation expert.

REFERENCES:

Adelsberger, H. H. and G. Neumann, "Goal Oriented Simulation Modeling Using PROLOG," Proceedings of the 1985 SCS Conference on Modeling and Simulation On Micro-Computers, San Diego, CA, pp:42-47, January 1985.

Adelsberger, H.H., U.W. Pooch, R.E. Shannon and G.N. Williams, "Rule Based, Object Oriented Simulation Systems," Intelligent Simulation Systems, SCS Simulation Series, Vol. 17, No. 1, pp: 107-112, January 1986.

Allen, E.M., R.H. Trigg and R.J. Wood, The Maryland Artificial Intelligence Group Franz LISP Environment, University of Maryland, November 1983.

Arons, H. de Swann, "Expert Systems in the Simulation Domain," Mathematics and Computers in Simulation, Vol. XXV, 1983.

Bartlett, F.C., Remembering, Cambridge University Press, Cambridge, MA, 1932.

Bobrow, D. and T. Winograd, "An Overview of KRL: A Knowledge Representation Language," Cognitive Science, Vol. 1, No. 1, pp:3-46, 1984.

Brazier, M.K. and R.E. Shannon, "Automatic Programming of AGVS Simulation Models," Proceedings of 1987 Winter Simulation Conference, Atlanta Ga, 1987.

Cleary, J.G. and A. Dewar, "Interpreters for Logic Programming - A Powerful Tool for Simulation," Proceedings of SCS Conference on Simulation in Strongly Typed Languages, San Diego, CA, February 1984.

Cleary, J., K-S. Goh, and B. Unger, "Discrete Event Simulation in Prolog," Proceedings of SCS Conference on Artificial Intelligence, Graphics and Simulation, San Diego, CA, pp:8-13, January 1985.

Dahl, O.J. and K. Nygaard, "SIMULA-An ALGOL Based Simulation Language," Communications of the ACM, Vol. 9, pp:671-678, 1966.

Ford, D.R. and B.J. Schroer, "An Expert Manufacturing Simulation System," Simulation, Vol. 48. no. 5, pp:193-200, May 1987.

Fox, M.S. and Y.V. Reddy, "Knowledge Representation in Organizational Modeling and Simulation: Definition and Interpretation," Proceedings of the 13th Annual Pittsburgh Conference on Modeling and Simualation, April 1982.

Futo, I. and J. Szeredi, " A Discrete Simulation System Based on Artificial Intelligence Methods," Discrete Simulation and Related Fields, (Ed. A. Javor), North-Holland, Amsterdam, pp:135-150, 1982.

Futo, I., "Combined Discrete/Continuous Modeling and Problem-solving," ECAI 84, (Ed. T. O'Shea), Elsevier Science Publishers, Amsterdam, 1984.

Gaines, B.R. and M.L.G. Shaw, "Expert Systems and Simulation," Proceedings of SCS Conference on Artificial Intelligence, Graphics and Simulation, San Diego, CA, pp:95-101, January 1985.

Goldberg, A., Smalltalk-80: The Interactive Programming Environment, Addison-Wesley, NY, 1984.

Heidorn, G.E., "English as a Very High Level Language for Simulation Programming," SIGPLAN Notices, Vol. 9, No. 4, pp:91-100, 1974.

Heidorn, G.E., "Automatic Programming Through Natural Language Dialog: A Survey," IBM J. Research and Development, Vol 20, pp:302-13, 1976.

Hill, T.R. and S.D. Roberts, "A Prototype Knowledge-Based Simulation Support System," Simulation, Vol 48, No.4, pp:152-161, April 1987.

Humphress, D. A., "Model Execution in a Goal-oriented Discrete Event Simulation Environment," a Ph.D. dissertation, Computer Science Dept., Texas A&M University, College Station, TX 1987.

Ketchem, M.G., G.L. Hogg and R.E. Shannon, "Memory Management, Data Structures, and the Development of Advanced Simulation Software in a Microcomputer Environment," Modeling and Simulation on Microcomputers, (Ed. R.G. Lavery), pp:67-70, SCS Publications, San Diego, CA 1985.

Ketchem, M.G., "Computer Simulation as a Decision Support Tool," a Ph.D. dissertation, Industrial Engineering Dept., Texas A&M University, College Station, TX, 1986.

Klahr, P., W.S. Faught, and G.R. Martins, "Rule-oriented Simulation," Proceedings of 1980 International Conference on Cybernetics and Society, IEEE, pp: 350-354, Cambridge, MA, October 1980.

Lenat, D., "AM: an AI Approach to Discovery in Math as Heuristic Search," a Ph.D. dissertation, Computer Science Dept., Stanford University, Stanford CA, 1976.

McArthur, D. and H.Sowizral, "An Object-oriented Language for Constructing Simulations," Proceedings of 7th International Conference on Artificial Intelligence, IJCAI, pp:809-814, Vancover, Canada, August 1981.

McArthur, D. and P. Klahr, "The ROSS Language Manual," RAND Corp. Report N-1854-AF, Santa Monica, CA, 1982.

Mellichamp, J.M. and A.F.A. Wahab, "An Expert System for FMS Design," Simulation, Vol.48, no.5, pp:201-208, May 1987.

Minsky, M., "A Framework for Representing Knowledge," The Psychology of Computer Vision, (Ed. P.H. Winston), McGraw-Hill, N.Y. 1975.

Murray, K.J., "Knowledge-based Model Construction: an Automatic Programming Approach to Simulation Modeling," a P.H.D. dissertation, Computer Science Dept., Texas A&M University, College Station, TX 1986.

Nyan, P.A., "A Comprehensive Environment to Object Oriented Simulation of Manufacturing Systems," Simulation in Computer Integrated Manufacturing, (Ed. K.E. Wichmann), European Simulation Multiconference, SCS Publications, pp:21-25, 1987.

Reddy, Y.V. and M.S. Fox, "Knowledge Representation in Organizational Modeling and Simulation: A Detailed Example," Proceedings of the 13th Annual Pittsburgh Conference on Modeling and Simulation, April 1982.

Reddy Y.V., M.S. Fox and N. Husain, "Automating the Analysis of Simulations in KBS," Proceedings of SCS Conference on Artificial Intelligence, Graphics and Simulation, San Diago, CA, pp:34-40, January 1985.

Salzberg, S., "Knowledge Representation in the Real World," AI Expert, Vol. 2, No. 8, pp:32-39, August 1987.

Seliger, G., B. Viehweger, B. Wieneke-Toutaoui and S.R. Kommana, "Knowledge-Based Simulation of Flexible Manufacturing Systems," Simulation in Computer Integrated Manufacturing, (Ed. K.E. Wichmann), European Simulation Multiconference, SCS Publications, pp:65-68, 1987.

Shannon, R.E., "AI Based Simulation Environments," Intelligent Simulation Environments, SCS Simulation Series, Vol. 17, No. 1, pp:150-156, January 1986.

Shannon, R.E., R. Mayer and H.H. Adelsberger, "Expert Systems and Simulation," Simulation, Vol. 44, No.6, pp:275-84, Jun. 1985.

SIMFACTORY With Animation, User's Manual, Version 1.3, CACI, Los Angles, CA, 1987.

Smith, Richard L. and Lucille Platt, "Benefits of Animation in the Simulation of a Machining and Assembly Line," Simulation, Vol.48, No. 1, pp:28-30, January 1987.

Ulgen, O.M., and T. Thomasma, " A Graphical Simulation System in Smalltalk-80," Simulation in Computer Integrated Manufacturing, (Ed. K.E. Wichmann), European Simulation Multiconference, SCS Publications, pp:53-58, 1987.

Wadhwa, S., C. Felix and J. Browne, "A Goal Directed Data Driven Simulator for FAS Design," paper presented at the European Simulation Multiconference, Vienna, 1987.

Zeigler, B.P., Theory of Modelling and Simulation, John Wiley & Sons, NY,1976.

AUTHOR'S BIOGRAPHY:

DR. ROBERT E. SHANNON is Professor of Industrial Engineering at Texas A & M University. His current research efforts are in the design of Fifth Generation Simulation Systems based upon AI/ES technology. He is the author of two books, *SYSTEMS SIMULATION: THE ART AND SCIENCE*, Prentice-Hall, (which won the H. B. Maynard Technical Book of the Year award and has been translated into Russian, Japanese and Farsi) and *ENGINEERING MANAGEMENT*, John Wiley & Sons, as well as over 50 journal and technical papers. Dr. Shannon is active in a number of Professional Societies including IIE (in which he is a Fellow), ORSA, TIMS, SCS and honorary societies such as Sigma Xi, Alpha Pi Mu and others.