Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Intelligent simulation environments: Identification of the basics

Jordan Snyder
Systems Engineer
Motorola Inc - GEG
Tempe, Arizona

and

Dr. Gerald T. Mackulack
Associate Professor of Engineering
Arizona State University
Tempe, Arizona

ABSTRACT

A problem exists in efficiently combining a non-deterministic decision capability with a current discrete event simulation language for use by the simulationist (programmer and in the future the user). This paper explores this problem in the context of the discrete event simulation problem domain implemented in Siman [tm]. The purpose is (1) to provide an ontological definition of abstract ideas from data to wisdom, (2) identify a taxonomy of simulation and artificial intelligence combination dialects, and (3) establish the need for and then introduce a "decide node" which will assist the simulationist in incorporating a broader spectrum of the ontology more easily than current dialects allow.

INTRODUCTION

Research and commercial software applications have been combining principles of artificial intelligence and simulation for many years. Attaining an efficient marriage of these paradigms will provide an environment which provides both qualitative and quantitative decision support. Taxonomies have been stipulated which have described the methods of combination (O'Keefe 1986). Environments have been developed and tested which investigate approaches for combination [(Reddy and Fox 1982) and (Umphress 1987)]. This paper investigates past taxonomies and current thrusts. Basic knowledge representations, their relative semantic definitions, and combination methodologies are also discussed. The objective is to discuss these taxonomies and methodologies relative to current and proposed simulation environments. The "decide" node is presented as an efficient means of providing the simulationist with the capability to model non-deterministic decisions.

The domain of discrete event simulation will bound this discussion. Our approach illustrates a situation where the activity points (ie., nodes or blocks depending on the particular software implementation) and process times may be known with some statistical significance, but decisions made at each activity point are not necessarily based on probability or conditional branching. These decision points, instead, will be based on abstractions based on non-quantitative concepts.

BACKGROUND

"Simulation is a tool by which a user defines a computerized representation of a real-world system and observes the behavior of the system as it progresses though time" (Banks and Carson 1984).

Not all real-world systems can be modeled in quantitative representations: this statement is based on experience and common sense. To represent more real world systems both quantitative and qualitative evaluators must be incorporated into a model.

Traditional simulation languages, such as Siman (Pegden 1985) and Slam II (Pritsker 1986), can provide adequate quantitative representation and analysis. These languages lack the capability to represent heuristic decision relations or algorithms in an easy manner. Traditional simulation languages allow predefined control mechanisms (ie., branch points, resource allocation/preemption, et al.). These low level control mechanisms, however, do not easily provide the user with the tools necessary to implement high level decisions. O'Keefe (1986) provides an interesting point, "a queue priority rule is knowledge". The semantic definition of items presented in this paper, Figure 1, prefers to classify queue priority rules and related items of predefined procedure as information. Information is static, hence it can be prescribed before model execution. Knowledge, however, is dynamic and therefore can not be instantiated until a certain state or condition(s) exist.

Pure artificial intelligence development languages start with the traditional symbolic environments; Prolog (Clocksin and Mellish 1984), Lisp (Winston and Horn 1984), and OPS5 (Brownston, et. al. 1985). Many simulation dialects have been built upon these languages. These dialects include ROSS (McArthur and Klahr:82), SIMKIT (Intellicorp 1985), and KBS (Reddy and Fox 1982) to reference just a few. These environments provide the flexibility to represent non-deterministic heuristic rules or statements. The current approach to marketing modern dialects is combining the core simulation code with user interface utilities. These utilities range from simple operating system calls to embedding word processing systems, control of peripherals and I/O ports, and debuggers. The addition of a graphical WIMP (widows, icons, menus, pull-downs/pop-ups) user interface with these utilities and

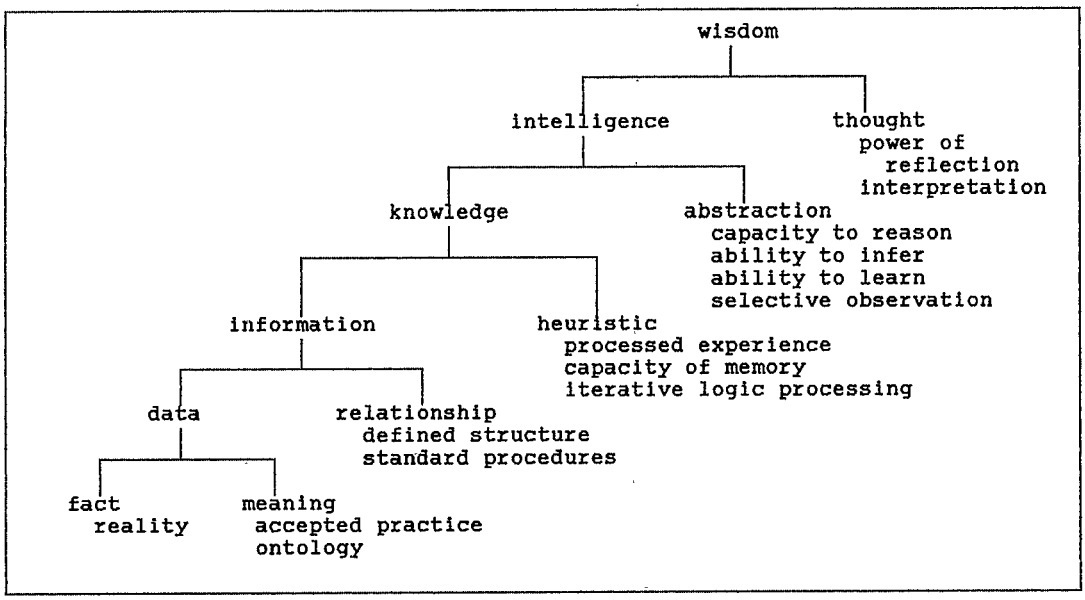

Figure 1: Hierarchical semantic definition of "Relative" knowledge along the continuum from fact to wisdom

the simulation dialect provides very flexible analysis environments. Balci and Nance (1987) provide an overview of some of the available simulation environments and discuss their requirements for a simulation model development environment. Their description of environments provide the basis for the current discussion of environments and the combination of symbolic processing and traditional simulation dialects.

When contrasting traditional simulation environments with pure symbolic environments we discover that both approaches have limitations. Traditional environments:

- lack direct ability to represent non-deterministic heuristics
- require the use of an event calendar to track universal time and schedule sequential events

Environments based solely on symbolic languages:

- usually require special hardware/software architectures
- may not guarantee optimal conclusions based on the traditional search algorithms usually employed in these dialects
- may not necessarily include the proper statistical analysis procedures to perform output analysis
- usually require the model representation and experimental data to be tightly coupled in the code

RESEARCH CONTRIBUTION

Ontological Identification

Strictly speaking, ontology refers to a branch of metaphysics relating the nature and relationships of being. Current literature discussing prototypes or implemented artificial intelligence based systems mention "knowledge bases". Databases contain data; what do knowledge bases contain - knowledge? Webster's Dictionary defines knowledge in many terms, ranging from cognition to the body of known truth. This definition does not satisfactorily identify what a knowledge base contains. This paper introduces a definition of knowledge in terms of a continuum based on the nature and relationships of being; an ontology.

Dan Appleton (1984) first introduced the idea of an ontological approach to the definition of datum. Dr. Appleton applies database analysis to datum and facts via 1:1 and 1:Many relationships to discern the difference between what is known and what is inferred. For example, "12345" may be considered a sequence of integers or a zip code depending on the context in which interpretation occurs. This approach was used in Figure 1 which illustrates a hierarchical definition of knowledge relative to the semantics of a continuum from fact to wisdom. For example, "data" is considered the result of applying meaning to fact ("12345" occurs on the front of an envelope). "Information" is the application of one or many relations among one or many pieces of data (the envelope is one of many given to a sorting machine used in a post office which is empirically studying

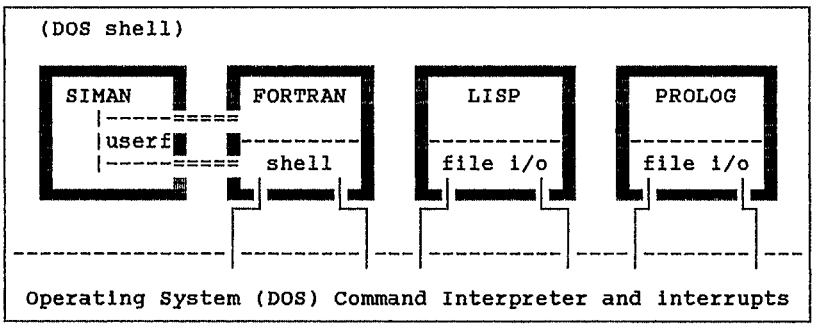

Figure 2: A multi-language platform used to investigate a heterogeneous mixture of qualitative evaluation and quantitative analysis

the efficiency of various sorting algorithms). As defined, knowledge is not just a rule, knowledge is the dynamic application of some heuristic to static information. Neither the heuristic nor information need be quantitative in nature. The application of this type of knowledge in a simulation is termed a "decision point". Traditional simulation languages do not easily provide the general user with the ability to incorporate these "decision points".

Combination Classification

The ensuing discussion provides descriptions of the three methods of combining symbolic processing techniques with current simulation methodologies. Symbolic processing techniques refer to artificial intelligence, knowledge base processing, heuristic search algorithms, expert systems, or wherever qualitative evaluations would be used.

Symbolic processing techniques and current simulation methodologies can be combined in three distinct groupings based on source code organization and interaction:

1. a homogeneous mixture
2. a separated, serial or pseudo-parallel, cooperation
3. a heterogenous mixture

Homogeneous Mixture:

A homogeneous mixture usually codes the simulation paradigm (including event scheduler/calendar for discrete event systems) in a symbolic (primarily qualitative) language. This approach relies on the symbolic language and its utilities to perform traditional simulation methods while having access to the flexibility of a symbolic language. Current approaches in the object oriented paradigm use the built-in messaging capability of symbolic languages to control the flow of data and information between the various entities (reference Figure 4 for a description of the five basic types of entities). Much research has addressed this approach [(Futo and Gergely 1987), (Rosenblit and Zeigler 1985), (Zeigler 1987), and (Reddy and Fox 1982)]. Some commercial packages are available which allow the implementation of frames and general symbolic processing [(Dahl and Nygaard 1966), (Goldberg and Robson 1983), and (Intellicorp 1985)].

Separated Cooperation:

The separate cooperation principle is the basis for intelligent front ends that act much the same way as fourth generation code generators. In most instances, symbolic-based code (AI/ES) interacts with the user to determine objectives, goals, performance criteria, and behaviors. The symbolic-based code then acts as a program generator, constructing the simulation model and experimental parameters in an existing quantitative-based simulation language.

The simulation is run and results are passed back to the symbolic environment, where relevant decisions regarding simulation continuation or control variable modification are made. This continues until the symbolic code (qualitative environment) is satisfied. The results are then passed back to the user in the form of recommended decision rules.

A very interesting twist (possibly a exception to this ontology) is provided by Flitman and Hurrion (1987). In their paper, "Linking Discrete-Event Simulation Models with Expert Systems", they describe the use of two separate microcomputer systems. One system supports a Fortran simulation language and model. The other system supports a Prolog simulation engine which is capable of monitoring and control (via parameter adjustment) the Fortran based simulation model. The two computer systems are linked via serial ports across a standard RS232C interface.

Heterogeneous Mixture:

The third combination methodology is a heterogeneous mixture at macro-code level. Ruiz-Mier and Talavage (1987) identify this type of approach in terms of a hybrid paradigm. Macro-code level involves tightly

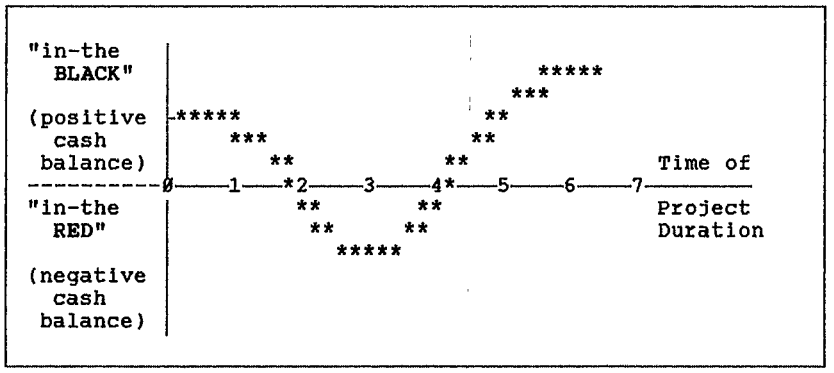

Figure 3: Example diagram of the changing interpretation of a qualitative evaluation -- the evaluation "GOOD".

coupled interaction of different languages. This allows the designer to use the tool (ie., language) best suited for the particular problem at hand. A tool currently unavailable is the "decide" node/block (node or block is dependent upon the simulation language paradigm; both descriptions are interchangeable for this particular discussion). This "decide" node should utilize qualitative analysis to assist in logic branching or decision points. Data, variables, and files created and maintained by the simulation may be accessed and modified by the symbolic-based code. Until a universal language paradigm and implementation becomes available which both efficiently and effectively combines qualitative evaluations and quantitative analysis, the heterogeneous mixture is the closest approximation.

PROPOSED SOLUTION

The problem of providing an easy method to include non-deterministic decision points still exists. Incorporation of the heterogeneous method with intelligent front and rear ends (the second method) would provide a modular and flexible environment with commercial applications. Efforts are currently being pursued within Arizona State University's Systems Simulation Lab to identify the applicability of a heterogeneous mixture using common dialects [(Cochran and Mackulak 1987), (Cochran et.al. 1987), and (Mackulak and Cochran 1987)]. An intelligent simulation environment is described by Hong et al. (1988). This environment uses standard tools currently available for the Intel 80x86 based microcomputers.

The environment includes:

- design goal formulation
- generic model selection
- experiment execution
- output interpretation and validation
- model/output documentation word processing
- various data/information/knowledge bases
  - -- user goals
  - -- statistics domain
  - -- historical results
  - -- error/debug codes

A subset research issue of this intelligent simulation environment is the driving force behind the identifications and classifications of this paper. Subsequent research results will describe the applicability of adding a non-monotonic "decide" node capability to an existing discrete event simulation language.

What capability is actually being added with a "non-monotonic decide node"? First, the application of non-monotonic logic must be understood. Elaine Rich's (1983) description is the basis for this discussion. In this particular case systems based on predicate logic constructs are monotonic in the sense that the number of statements known to be true is strictly increasing over time. New statements can be added to the system and new theorems can be proved, but neither of these events will ever cause a previously known or proven statement to become invalid. The addition of a piece of information which forces the deletion of a previous belief is a non-monotonic reasoning process. Consider the following example:

A company starts a project from ground-zero with a bucket of money. In the beginning, the money is spent on research and development during which time no profits are realized. From an accountants view the project is still in the black, but decreasing (Figure 3).

At time 0 the project could be evaluated as "GOOD" based on the financial criteria of positive cash balance. As the project proceeds, the beliefs which form the basis for financial criteria change (non-monotonically). At time period 3, the project may still be evaluated "GOOD" because the fact that further capital outlay is not necessary and a return on investment is expected shortly. Although at period 3 the direct evaluation of the financial criteria yields a negative cash balance, this evaluation must also be

considered within context of the overall system.

This example illustrates the use of qualitative non-monotonic decision ability. As long as the financial evaluation is "GOOD", the project continues. However, as the project continues the beliefs which support the financial evaluation change, affecting previous beliefs and assumptions.

The SIMAN [tm] language will be used as the basis for the quantitative simulation environment. An existing simulation environment will be used because it is:

1. currently available
2. optimized
   a) algorithmically from the experience of the programming company and its consulting experience
   b) compiled for high machine efficiency in a specific hardware configuration
3. proven to work if used properly (hence validation efforts can focus on the modifications resulting from a hybrid operation with the embedded symbolic language)

Lisp and Prolog will provide the symbolic processing capabilities for this investigation. Most previous work has been coded in Lisp or Prolog, therefore collected expertise in the form of dialect specific knowledge representation can be found in the literature. The user must understand the five types of basic knowledge necessary to completely model a real-world system. A frame based approach similar to the object oriented paradigm will be used to simplify user understanding.

The five types of basic knowledge necessary to completely model a real-world system are identified in Figure 4. The "objects" referred in this definition may be:

1. physical entities with attributes
2. rules with instantiated variables
3. structure of active intellect with at least one unbound variable

The specific outcome targeted is a frame-based qualitative non-monotonic decision ability added to an existing general purpose discrete event simulation language. This ability will be provided as a "decide" node added to the simulationist's tool box. In the previous example, the decide node would have been used to monitor the system from a financial viewpoint. If the system was evaluated as "GOOD" no alterations would occur. However, at some time if the qualitative financial evaluation was "not-GOOD" a flag would be raised, and further processing could occur.

The generic "decide" node will:

A. make assumptions based on the
   -- current global state of the system
   -- overall system performance
   -- trend of global system
   -- trend of components or modules of system

B. revise the "belief-set" based on assumptions (and their applicability) and reasoned beliefs

C. allow decisions to affect the five basic types of knowledge listed in Figure 4

Representation entity	Representation use
1. system	{the state}
2. object$_a$	{self rule}
3. object$_a$ <==> system	{stimulus and behavior}
4. object$_i$ <==> object$_j$	{interaction}
5. simulation objectives and/or experimental goals	{reason for existence}

Figure 4: Simulation entities and the type of knowledge construct they represent

Point A is consistent with the spirit of a closed-loop simulation environment. Point B is a derivative of Doyle's work (Doyle:79) on applied non-monotonic logic encompassed in a "Truth Maintenance System". Filman (1988) has investigated this approach within the context of IntelliCorp's KEE [tm] language. Although this attempt is not generic, the approach has proven to be beneficial. Point C is an extension of the object oriented approach to control via messages. Execution is not dependent on messaging (the execution is still driven by the traditional simulation engine), but parameter adjustment can occur as a result of pseudo-message passing. The "decide" node is a system/entity monitor capable of altering the control of an execution based on assumptions, beliefs, and knowledge application.

CONCLUSION

The problem of efficiently combining a non-deterministic decision capability to a current discrete event simulation language can be solved by using the "decide" node. This node is based on the need to represent changing qualitative evaluators. The "knowledge" behind the evaluators has been identified, in relative terms, in Figure 1. We have stipulated three separate structures and combination methodologies (classifications) based on source code interaction. Using these identifications of knowledge and classifications a solution environment currently being pursued at Arizona State University's Systems Simulation Lab has been described. Future research will provide

insight into the applicability of the "decide" node and performance of the proposed implementation.

ACKNOWLEDGEMENTS

Kolseth, Camala, McDonnell Douglas Helicopter Company.

Krecker, Elizabeth, Skyword Marketing.

REFERENCES

Appleton, Daniel S. (1984), "Business Rules: The Missing Link," Datamation, October 15th, pp 145-150.

Balci, Osman and Nance, Richard A. (1987), "Simulation Model Development Environments: A Research Prototype", Journal of Operations Research Society, V38:N8, pp 753-763.

Banks, Jerry and Carson, John S. (1984), "Discrete-Event System Simulation," Prentice-Hall, Englewood Cliffs, New Jersey.

Brownston, L., Farrell, R., Kant, E., and Martin, N. (1985), "Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming," Addison-Wesley, Reading, Massachusetts.

Clocksin, W.F., and Mellish, C.S. (1984), "Programming in Prolog," Second Edition, Springer-Verlag, Berlin, Heidelberg.

Cochran, J.K., and Mackulak, G.T. (1987), "Low Cost Artificial Intelligence Can Aid Design and Simulation," CIM Review, V3:N3, pp 33-36.

Cochran, J.K., Mackulak, G.T., Castillo, D., and Du, E. (1987 Jan), "Configuring Available Software Into an AI/ES Environment for Automated Manufacturing Simulation Design on the PC," SCS Conference on Simulation of Computer Integrated Manufacturing Systems and Robotics, San Diego, California, pp 1-7.

Dahl, O.J., and Nygaard, K. (1966 Sep), "SIMULA - an ALGOL-based Simulation Language," Communications of the ACM, V9:N9, pp 671-678.

Doyle, Jon (1979), "A Truth Maintenance System", Artificial Intelligence, V12:N3, pp 231-272.

Filman, Robert E. (1988 Apr), "Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environment", Communications of the ACM, V31:N4, pp 382-401.

Futo, Ivan, and Gergely, Tamas (1986 Jul), "Logic Programming in Simulation," Transactions of the Society for Computer Simulation, V3:N3, pp 195-216.

Goldberg, A., and Robson, D. (1983), "Smalltalk-80: The Language and its Implementation," Addison-Wesley, Reading Mass.

Hong, Suck-Chul, Cochran, J., and Mackulak, G. (1988), "Specification of an Architecture for Intelligent Simulation Environments", Arizona State University Systems Simulation Lab (available in the 1989 Multi-Conference Proceedings).

Intellicorp (1985), SIMKIT (tm) User's Manual, Intellicorp, Inc., Melno Park, California.

McArthur, David, and Klahr, Philip (1982), "The ROSS Language Manual," Report N-1854-AF, Rand Corp., Santa Monica, California.

O'Keefe, Robert (1986 Jan), "Simulation and Expert Systems - A taxonomy and some examples," Simulation, 46:1, pp 10-16.

Pegden, C. Dennis (1985 Jul), "Introduction to Siman with Version 3.0 Enhancements," Systems Modeling Corporation, State College, Pennsylvania.

Pritsker, A. Alan B. (1986), "Introduction to Simulation and SLAM II," Third Edition, Halsted Press, New York.

Reddy, Y.V., and Fox, Mark S. (1982 Sep), "KBS: An Artificial Intelligence Approach to Flexible Simulation," Technical Report, Carnegie-Mellon University, Robotics Institute, CMU-RI-TR-82-1 SDL #495.

Rich, Elaine (1983), "Artificial Intelligence," McGraw-Hill, New York.

Rosenblit, Jerzy W., and Zeigler, Bernard P. (1985), "Concepts for Knowledge-based System Design Environments," Proceedings of the 1985 Winter Simulation Conference, pp 223-231.

Ruiz-Mier, Sergio, and Talavage, Joseph (1987 Apr), "A hybrid paradigm for modeling of complex systems," Simulation, 48:4, pp 135-141.

Umphress, David Ashley (1987), "Model Execution in a Goal-Oriented Discrete Event Simulation Environment," Phd Dissertation, Texas A&M University.

Winston, Patrick Henry and Horn, Berthold (1984), "Lisp", Second Edition, Addison-Wesley, Reading Massachusetts.

Zeigler, Bernard P. (1987 Nov), "Hierarchical, modular discrete-event modelling in an object-oriented environment," Simulation, 49:5, pp 219-230.

AUTHOR BIBLIOGRAPHIES

Jordan Snyder

Industrial & Management Systems Engineering
College of Engineering
Arizona State University
Tempe, Arizona 85287
(602)965-3185

Jordan Snyder is a graduate student in the School of Industrial Engineering and Management Sciences at Arizona State University (ASU) and a Manufacturing Systems Engineer at Motorola Inc. He received his Bachelors degree in Mechanical Engineering from ASU. Jordan is currently a member of the System Simulation Laboratory. His overall research interest is the application of artificial intelligence algorithms to traditional simulation languages in an attempt to decrease the knowledge/experience necessary to construct and execute a model. Jordan's primary thrust is the application of reasoning/belief algorithms to aid in qualitative decision processing.

Gerald T. Mackulak, Phd.

Industrial & Management Systems Engineering
College of Engineering
Arizona State University
Tempe, Arizona 85287
(602)965-6904

Dr. Mackulak is currently an Associate Professor of Engineering in the Department of industrial and Management Systems Engineering at Arizona State University (ASU). He is also Director of the Systems Simulation Laboratory (SSL), a newly created laboratory within the CIM Systems Research Center. The SSL has as its' charter the development of new simulation tools and methodologies, with specific concentration in the areas of expert systems and statistical post-processing. Dr. Mackulak received all his degrees from Purdue University in the area of Industrial Engineering. Before joining ASU eight years ago he held positions with U.S. Steel, Burroughs, and the simulation consulting firm of Pritsker and Associates. Dr. Mackulak has published extensively in the area of CIM integration methods and has taught both public and private seminars on CIM system integration, simulation and economic justification of CIM implementations. His current research is centered around developing expert systems that enable even novice users to effectively use simulation.