Technical Report CS81003-R

MODEL REPRESENTATION IN DISCRETE EVENT SIMULATION: THE CONICAL METHODOLOGY*

Richard E. Nance
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

15 March 1981

---

* Discussions and correspondence with C. Michael Overstreet, John S. Laski, Stephen C. Mathewson, and Bernard P. Zeigler are acknowledged as contributing to ideas developed in this paper. The author is grateful to the Naval Surface Weapons Center for providing an exposure to simulation model development for combat systems design and test during September 1979 - May 1980. The author is also grateful to Mrs. Donna K. Seymour for her excellent typing of this manuscript.

Abstract

Beginning with a brief review and classification of model development approaches, we characterize the simulation model life cycle as comprised of seven phases: the conceptual model, the communicative model, the programmed model, the experimental model, model results, use of the model for integrated decision support, and the modification and extension of of the model. This characterization places severe requirements on the task of model management (creation, acceptance, use, revision or extension, and reuse). The Conical Methodology has been developed in response to the needs that predominate the model development phases (from conceptual model to model results). Definitions used in the Conical Methodology are explained, and the approach is illustrated with a machine repairman example. An incomplete critique of the result and the approach concludes the paper.

I. Simulation Model Development

Simulation model development is the process by which a modeler (or modeling team) organizes a conceptual model for solving some problem, transfers it into a communicative form that admits to solution using simulation, and uses the model results, albeit with modifications to the original model, for useful problem-solving purposes. The term "model development" as used herein encompasses the creation, testing (with validation and verification), and use (or implementation) of a model. While "simulation" has different meanings, we view the term as encompassing three related problem-solving techniques:

> Monte Carlo simulation - problem solution by subjecting a model to a repetition of statistical trials, which are analyzed primarily by classical statistical techniques (assuming independent, identically distributed outcomes),
>
> continuous simulation - experimentation with a model that progresses from state to state continuously without explicit state recognition (or discontinuities), typically represented as a set of differential equations, and
>
> discrete event simulation - experimentation with a model that progresses explicitly from state to state in uneven time increments (discontinuously), typically exemplified by queueing models.

Closely related to simulation are the areas of gaming (or "operational gaming"), in which a simulation model serves to create an artificial environment interacting with a human, and emulation, in which basic attributes of the model are possessed by the computer system on

which the model is placed, i.e. the computer system itself assumes an essential modeling role. Note that simulation does not mandate the use of a digital computer, but the practical problems of today can only be solved in this way.

While fundamental similarities exist among the three simulation techniques, the differences are not inconsequential. In particular, the role of differential equations as a general model representation language for continuous models has no counterpart in the discrete event domain. This paper addresses the development of discrete event models, recognizing that the concepts and approach have validity and utility for the other domains as well. However at this juncture the creation of a methodology for discrete event simulation, and its recognition and acceptance by that community, appears sufficiently challenging without struggling with the resolution of "pathological" distinctions.

The objectives of this paper are:

(1) to review the work in simulation model development, categorizing it by approach,

(2) to explain the need met by the Conical Methodology (CM) by comparing its objectives with those of other simulation model development approaches,

(3) to describe the Conical Methodology (CM) and present the definitions forming its foundation,

(4) to illustrate the use of CM in developing a model of a machine repairman problem,

(5) to critique, although incompletely, both the CM and the model representation produced by it, and

(6) to relate the role of model development, and the CM approach to model development, to the model life cycle.

A summary, noting some implications of the CM for simulation model development, concludes the paper. The organization of the paper parallels the division of the objectives.

I.1 Importance of the Development Process

Surveys of the use of problem-solving techniques in operations research show that simulation persists as a highly used technique [SHANR70, SHANR80]. Yet, simulation shares the same problems, and consequently the same criticisms, applicable to other techniques requiring the creation of large software systems. The dramatic increase in computational capabilities has stimulated ever-increasing requirements for simulation studies. The creation, validation,

verification, experimentation, maintenance, and modification of simulation models is now a costly, time-consuming effort. One estimate for a large defense project, which is not to be named, places the cost of simulation software at ten million dollars over a six year period.

In 1976 the GAO cited several problems that led to high costs and limited utility of "computerized models" in the federal government [USGAO76]. Studies sponsored by the National Bureau of Standards during the same time frame were investigating the feasibility of model documentation standards for computerized models in general [CONTA77, ROTHP78] and simulation models in particular [NANCR77b]. A model evaluation project by the Department of Commerce sought to establish guidelines by which computerized models could be assessed as to their acceptability.

On the international scene a workshop held in September, 1979 explored the issues of standardization of simulation languages. Although located in St. Agata, Italy (near Sorrento), the workship was labeled SWISSL (the Sorrento Workshop on International Standardization of Simulation Languages). Several views were put forth, and an organizing body, the Committee for International Standardization of Model Oriented Languages (CISMOL), was formed. Also on the international scene, the Technical Committee on Simulation Software (TC3) of the International Association for Mathematics and Computers in Simulation (IMACS) is addressing the issue of including discrete event and combined models in the revision of its language standard for continuous models, adopted initially in 1967 [SCICS67].

The large investment represented by a simulation model and the breadth of concern for improvements in the application of simulation attest to the significance of simulation model development. However, the suggestion of standardization as a solution route has drawn cautionary responses [NANCR79, SOLOS80]. Recently, we have argued that a major contributor to the high cost of simulation is the absence of fundamental principles, caused in part by a theory/interpretation inversion induced by the focus on simulation programming languages (SPLs) [NANCR81]. Only by establishing an orderly base — a set of sound, consistent definitions — can a broadly applicable, more axiomatic model development methodology be created. This "retreat to first principles" is a precursor to the development of meaningful and useful guidelines, if not standards, for simulation model development.

I.2 Related Work

An early attempt to generalize simulation model representation is the calculus of change concept, proposed by Lackner [LACKM62, LACKM64a, LACKM64b]. An explanation of this work and a conjecture regarding its inability to influence model representation are given in a previous paper [NANCR79].

In reviewing the ongoing work in simulation model development, we have organized the efforts into four categories:

(1) the extension of software development tools,

(2) program generators,

(3) simulatin theory, and

(4) systems specification languages.

I.2.1 The Extension of Software Development Tools

A concern for inadequate and inconsistent model documentation caused McLeod to construct a programmer's checklist [MCLEJ70, MCLEJ73]. McLeod's objective was not simply to ease the documentation task but to establish some uniformity and completeness in the documented information. Earlier efforts to ease the documentation task, e.g. the automatic production of GPSS flow diagrams from the program, proved of little benefit. More recent aids such as programs for variable cross referencing in SIMSCRIPT II.5 and the recognition of all class attributes in SIMULA [VISOG79] offer more assistance in model creation.

A more inclusive approach to model development through the extension of software design tools is the Software Design and Documentation Language (SDDL) created by Kleine [KLEIH77a]. SDDL is an interactive system which assists in the design of SIMSCRIPT programs. The utility of SDDL for simulation program development, especially the production of complete documentation, is described in a subsequent paper [KLEIH77b]. The recent paper by Richerson [RICHM80] also contains helpful development suggestions for large FORTRAN simulation efforts.

In contrast with SDDL and the ad hoc, although quite helpful, techniques of Richerson, both of which initiated from program design and development, the cellular simulation project of the Centre in Simulation at the University of Lancaster began as a structured approach to developing large simulation models. The approach partitions a model into cells, which communicate by message transmissions. The relative independence of cells permits their parallel design and creation [DECAR76]. Recent extensions of the cellular structure propose that each cell can be viewed as a further cellular partition based on different aspects. For example, a cellular partition of a machine shop could be viewed as comprised of an "operational" cell (all machines running as normally expected) and a "degraded mode" cell (the optional structure of the shop if one or more machines are not operating normally). These cellular models of different system aspects could be developed almost independently as are the original partitions. The hierarchical structuring of the cellular partitioning also enables an efficient handling of the activity-based timing mechanism by a modification to the common three-phased approach [CROOJ80].

I.2.2 Program Generators

A program generator is a program that produces a program in a simulation programming language (SPL) from a simple input description. Tie program generator functions as a preprocessor facilitating the production of a skeletal structure of the model programmed in a "target"

SPL. Program generators can be designed as pedagogical tools as well, permitting relatively unsophisticated users (of SPLs) to produce simulation programs that can illustrate modeling design and analysis concepts [MATHS80].

The first program generator was the "Programming by Questionnaire" (PBQ) system developed at RAND [OLDFP66, OLDFP67]. However, the concept of interactive program generators for simulation models was first expressed by Clementson [CLEMA73]. Other than the work of Heidorn [HEIDG74], which sought to enable the natural language description of GPSS models, research and development of simulation program generators has been concentrated in Europe: CAPS/ECSL at Birmingham by Clementson [CLEMA73], DRAFT at Imperial College by Methewson [MATHS74, MATHS75], S4 at Sheffield by Lafferty [LAFFH73), MISDES by Davies at Portsmouth (now at Munich) [DAVIN76], and GASSNOL at Université Paul Sabatier by Vidallon [VIDAC80].

A common feature of all the simulation program generators developed in the UK is the reliance on the entity cycle (or activity cycle) diagram, originated as the "wheel chart" by Tocher [TOCHK64] and popularized in HOCUS [HILLR69], as the input language. (A good explanation of entity cycle diagrams is given in [POOLT77].) The simplicity of these diagrams, which use only four symbols, makes them easy to teach, and students quickly master the approach. The GASSNOL systems uses the Network Oriented CAD Input Language (NOCADIL), which is a command language designed around the description of nodes and arcs comprising a network.

I.2.3 Simulation Theory

The calculus of change concept cited above represents an attempt to derive fundamental modeling abstractions for discrete event simulation. Other attempts to identify some general concepts for model representation are found in [BLUNG67, NANCR72, and ETSCM72]. However, the most extensive formulation of a theory has utilized the concepts of general systems theory.

Zeigler's book [ZEIGB76], which draws together previous results published in several papers, presents a formalism that maintains a separation of static and dynamic model description. Model components are characterized by descriptive variables, and a components interaction section prescribes the dynamic relationships through an informal description. Additional dynamic description is furnished by "influencer diagrams" and "model trajectories." At the nucleus of the model representation is a finite state machine description. Recently, Zeigler's formalism and his elaborated development of the theory of experimental frames have permitted some important characterizations of models and the relations between models and systems [ORENT79].

Another systems-theoretic approach to simulation theory is given in a series of papers by Kindler [KINDE76, KINDE77, KINDE79]. Kindler's treatment also separates the static and dynamic aspects, producing a theoretical structure through a convergence from the static relationships among systems and models to the dynamic one. His approach has also been used to develop a classification of SPLs [KINDE78].

A different path to simulation theory has been followed by Harry Markowitz, one of the founders of SIMSCRIPT. In a lengthy and very interesting report, Markowitz reexamines the entity, attribute, and set (EAS) status view of SIMSCRIPT, noting its applicability as a generalizing description, applicable to simulation modeling and data base management systems [MARKH77]. Some definitional changes in SIMSCRIPT II are required to accommodate the EAS view, and Markowitz identifies them as well as explaining the original intent for seven levels of the language (only five are defined in [KIVIP73]) and the utility of the added levels. The generality of the EAS status view is also proposed as sufficient for its use as a standard for simulation model representation in a subsequent report presented at the SWISSL, mentioned above [MARKH79].

I.2.4 Systems Specification Languages

A systems specification language (SSL) contans elements of the objectives of a program generator. However, a SSL includes the function of the input language (the entity cycle diagram of the program generators) and seeks to produce a complete executable system specification (rather than a skeleton). The DELTA Project, a joint effort of the Norwegian Computing Center and the University of Aarhus (Sweden), is producing a SSL for discrete event simulation applications [HOLBA77]. (This is the lone SSL effort in simulation, although, Heidorn's natural language specification of GPSS models, now inactive,

shares similar objectives.) A SSL should permit a user from a known application domain to construct an executable simulation program using the terminology common to that application domain. The structure of DELTA reflects the SIMULA influence, which is to be expected. It also exhibits proper top-down design with stepwise refinement.

Beginning as a system development tool only, DELTA is now intended to produce executable code and a model specification in three related language forms: DELTA, BETA, and GAMMA. To date we are not aware of an application of DELTA to develop an actual simulation model.

I.3 Summary

Each of the above research and development areas influences the task of developing simulation models. While systems theory provides a descriptive theory of simulation experimentation, program generators demonstrate a utilitarian "proof by example." Software development tools are supported for model development by their acceptance in the programming community. Systems specification languages represent an ambitious extension of the current limited ability of the program generators.

Each area has its proponents, and each can rightfully claim successes in reducing the magnitude of the model development task. However, none appears to be a supportive tool throughout all phases of

model development. None offers both a descriptive and instructive approach to simulation model development. The foundation promoting an axiomatic structure supporting the entire model life cycle, remains to be established [NANCR81].

The remainder of this paper describes the Conical Methodology (CM) -- a conceptual tool for model development. We begin with defining the context in which large simulation models are created and used so as to explain the objectives of the CM (Section II). In Section III crucial definitions are presented and illustrated with a machine repairman example. Application of the methodology is described in Section IV. Section V contains an incomplete critique of the CM; termed "incomplete" because this conceptual tool can be evaluated only through an implementation test.

II. The Conical Methodology: Needs and Objectives

II.1 Phases in the Development of a Simulation Model

We perceive the use of a simulation model, its life cycle in the software development terminology, to consist of seven phases. These phases begin with a conceptual model, derived from the modeler's knowledge and assumptions about the modeled system and conclude with a phase in which the results of multiple experiments are categorized and

analyzed so as to permit knowledge-based experimentation. These phase definitions and the relationships among phases are presented in Figure 1.

THE SYSTEM

Either physical or real, the system and the study objectives provide the reference for the model and the modeling task.

PHASE 1: CONCEPTUAL MODEL

The conceptual model is that model which exists in the mind of the modeler. The form of the conceptual model is influenced by the system, the perceptions of the system held by the modeler (which are affected by the modeler's background and experience and those external factors affecting the particular modeling task), and the objectives of the study.

PHASE 2: COMMUNICATIVE MODEL

A model representation which can be communicated to other humans can be judged or compared against the system and the study objectives by more than one human. Several communicative models could be constructed during a study, each derived from a preceding communicative model (following the first) or different conceptual models. Entity cycle diagrams are examples of communicative models.

PHASE 3: PROGRAMMED MODEL

A programmed model is a model representation that admits execution by a computer to produce simulation results. It is a communicative model from which experimental results are obtained. SIMSCRIPT or SIMULA programs are examples of programmed models.

PHASE 4: EXPERIMENTAL MODEL

The programmed model and the executable description of the test environment (the experimental frame of Zeigler [ZEIGB76]) form the experimental model.

PHASE 5: MODEL RESULTS

The results phase includes the outcome from a single execution of the experimental model or those results produced to satisfy a single test scenario, which might require several model executions with different input value specifications, structural changes, etc.

PHASE 6: INTEGRATED DECISION SUPPORT

Integrated decision support extends the experimental domain to multiple scenario executions, indexed and accessible either by automatic, manual, or combined analysis so as to permit the recognition of behavioral features or trends unspecified in the individual tests. The recognition of untested but interesting test scenarios is possible. Analysis permits the extrapolation or prediction of untested scenarios based on prior results.

PHASE 7: MODIFIED MODEL

Modification represents a significant change in the model definition and specification from the original. The change might be caused by an extension of function of the model or restatement of the study objectives.

Figure 1. The Phases of Simulation Model Development

While many simulation models have been developed without explicity recognition of all of these seven phases, we believe that most successful models and modeling projects have experienced the first five phases. As the complexity and size of the model (and project) increases, the greater the necessity for explicitly recognizing each phase. The six and seventh phases could be applicable only to those projects in which a model is expected to have an extended life and be subjected to progressive experimentation with probable modifications. The programmed model (third phase) is what Lehman [LEHMM80, p. 4-8] has described as a P-program (real world problem solution):

> The problem statement can now, in general, no longer be precise. It is a model of an abstraction of a real-world situation, containing uncertainties, unknowns, arbitrary criteria, continuous variables. To some extent it must reflect the personal viewpoint of the analyst. Both the problem statement and its solution approximate the real world situation.

Lehman's comments on the life of a P-program could be applied verbatim to that of a large complex simulation model [LEHMM80, p. 8].

> Dissatisfaction will arise not only because information received from the program is incomplete or incorrect, or because the original model was less than perfect. These are imperfections that can be overcome given time and care. But the world too changes and such changes result in additional change. Thus P-programs are very likely to undergo never-ending change or to become steadily less and less effective and cost-effective.

These observations also support the existence of phases six and seven of the model life cycle.

One final comment on the simulation life cycle involves documentation, both program and model documentation. All too often documentation is perceived as an intervening phase between three and four, i.e. completion of the programmed model is followed by production of the "documented model" illustrated in Figure 2.

Program documentation is not to be done as an "after the fact" completion task. The proper role of program documentation is presented in numerous books and papers on software engineering. We take the position that model documentation demands far more than simply program documentation. Further, proper model documentation initiates with the first communicative model and continues through succeeding model development phases. This position and the relationship between model specification and documentation are detailed in previous papers [NANCR77b, NANCR79].

THE SYSTEM

CONCEPTUAL MODEL

COMMUNICATIVE MODEL

PROGRAMMED MODEL

COMPLETED (DOCUMENTED) MODEL

The completed model includes the necessary documentation, both internal to the program and as external documents, in order to use the program for experimental purposes.

EXPERIMENTAL MODEL

MODEL RESULTS

INTEGRATED DECISION SUPPORT

MODIFIED MODEL

Figure 2. Common and Incorrect Perception of the Documentation Role

II.2 Model Development Needs

Oren and Zeigler [ORENT79] have identified concepts needed for coping with the large complex simulation models of today. The inadequacies of current tools are identified by referring to the functional elements of a simulation program. We find much in this work that reinforces our own observations, and the experimental frame concept, originally presented in [ZEIGB76], provides an instructive guide for validation and verification. However, we do not share the authors' conviction that simulation programming languages based on systems theory can provide the form for most convenient expression [ORENT79, p. 70].

Recently, Oren has described Computer-Aided Modeling Systems (CAMS), which takes a broad view of model development, similar in breadth to our own [ORENT80]. He categorizes the needs for computer assistance in the modeling process as: (1) model generation/referencing, (2) model acceptability, (3) model processing, and (4) behavior processing. Oren's categories span the first five phases specified above, and his inclusion of adaptive or learning systems as part of behavior processing could also include the last two phases. However, we have difficulty in integrating his four categories into the requirements of model life cycle support or the more limited scope of model development.

Section I reviews several approaches to simulation model development. The contributions of current simulation model development approaches to each of the seven phases is subjectively assessed in Figure 3. Each reader must reach a conclusion regarding the accuracy of these assessments. However, we believe that the relative judgments are explainable and defensible.

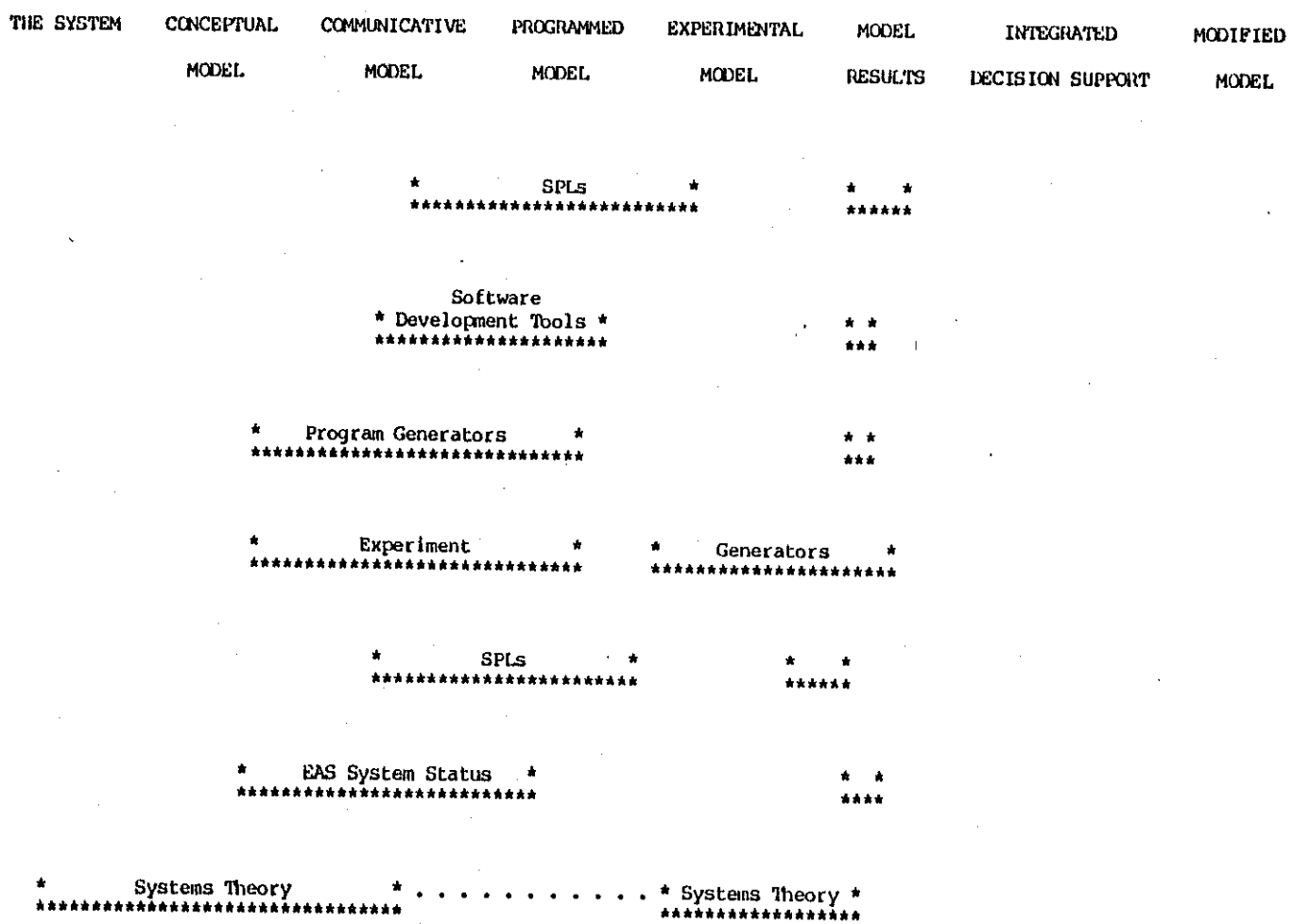

Figure 3. Comparative Contributions of the Approaches to the Model Development Phases

To provide a sketchy explanation, we offer the opinion that:

> SPLs provide less toward guiding the construction of a communicative tool than a software development tool because they are limited by the syntax and semantics of the language's world view as well as potential implementation restrictions. However, the SPL provides access to model results using system-defined variables for instance that cannot be matched by other approaches.
>
> The experiment generators extend the objectives of the program generators to the entire experimental environment. In so doing they should assist the user in formulating the experimental design, constructing the test scenarios, and formatting the model results.
>
> Markowitz's EAS approach contributes more than software development tools to the conceptual model, but its restriction to an event scheduling world view restricts the programmed and experimental models more than the program generators approach.
>
> The dotted line segment separating the systems theory contributions is intended to represent the indirect and uncertain contributions to the communicative and programmed models. This assessment is based on the required knowledge of systems theory (set theory and functional notation) to comprehend the communicative model. Production of a programmed model from the communicative model is not accomplished as simply as with program generators for example. The experimental frame concept contributes significantly to the experimental model, and the systems theory approach provides the only linkage between system and conceptual model (all others presume a suitable conceptual model as an initial condition).

The most apparent observation to be made from Figure 3 is the absence of contributions to the integrated decision support and the modified model. These two phases do not represent the fundamental challenges of model representation but do place crucial demands on the preceding phases of the model life cycle.

II.3 Objectives of the Conical Methodology

The Conical Methodology is a model development approach. It is not intended to provide abstract or concrete definitions of general systems. Moreover, it is limited to dealing with the relationships among models or, more precisely, model representations. It does not provide a solution to the very important problem of model validation -- accepting the model as a "truthful" representation of the system considering the objectives of the study. The CM is intended to contribute toward model verification -- the assurance that a model meets the stipulated requirements, i.e. the accurate translation from one representation to another.

With regard to the model life cycle, the Conical Methodology is intended to contribute to the first through the fifth phases -- the phases that relate to the model development. Specifically, the CM has the following objectives:

(1) assist the modeler in structuring and organizing the conceptual model,

(2) impose an axiomatic development within an apparently free and unrestrictive model creation system,

(3) utilize model diagnostics to assess measures such as completeness and consistency for verification purposes and relative model complexity for planning purposes,

(4) produce major model documentation throughout the model development as an essential byproduct of definition and specification but permitting the modeler to expand the descriptions as deemed necessary, and

(5) promote an organized experimental design and monitor the realization of that design in the experimental model.

These objectives are quite ambitious, but the simulation applications of today, not to mention those of tomorrow, demand no less.

III. A Limited View of Simulation Model Development

The development of a model is a translation effort. Any translation effort can be characterized simply as a process that, for some input, produces an output. The output of model development is a model representation that permits communication of the model characteristics and experimentation with behavior believed to describe the behavior of the modeled system. The inputs for model development could take various forms for example block diagrams, flowcharts, entity cycle diagrams, or a mental perception of the model. We take the view that model development can lead to several evolutionary representations, but the mental perception is the initial representation for every component.

Model development begins with a set of definitions that permit an accurate, unambiguous, and complete description consonant with the objectives of the simulation study. Also required is a structure by which the parts of a model can be related, i.e. model development tools should provide a discipline for model composition and decomposition. We contend that the development task should provide a large part of the model documentation and that diagnostic assistance to the modeler should proceed throughout the model development. The objective of this paper is to deal with the first and most basic element of simulation model development: an adequate, descriptive, and conducive set of definitions.

We presume that the motivating purpose for model development is in order to experiment with the model. This experimentation is accomplished through model execution or, most likely, repeated executions of the model. Implementation of experimental results is assumed to extend over a lengthly period so that modifications and extensions to the model are quite likely.

III.1 The Machine Repairman Example

To assist in the explanation of definitions, we use the classical machine repairman model (see, for example, [PALMD47, COXDR62].

> A single repairman is assigned to service a group of semiautomatic machines which fail at random points in time. The repairman can service a failed machine in a time period that is assumed to be negative exponentially distributed with parameter MU. On completing a machine

service, the repairman determines if any machines are failed; if so, a failed machine is selected for repair. If all machines are operating at the conclusion of a repair, the repairman walks to a designated neutral location to await the next failure. The walk times, to machines and to the neutral location, are functionally determined by the identifiers of the two locations. The machine failures are assumed to follow a Poisson distribution with parameter LAMBDA.

Note that the selection of a machine to be repaired, if more than one are failed, is intentionally left undefined. The repair selection distinguishes among three variations of the problem; the first failed/first repaired (first come/first served) discipline is used in a following section.

III.2 The Definitions

We utilize two conventions in presenting the definitions: (1) the use of indentation to reflect subordination among terms, and (2) the underscoring of a term used in the definition which is subsequently to be defined. References to the machine repairman example are included at various points for clarification.

A model of a system is comprised of objects and the relationships among objects.

An object is anything that can be characterized by one or more attributes to which values are assigned.

Attributes are object descriptors of two types: indicative and relational.

Indicative attributes describe an aspect of the object; i.e. provide some knowledge about the object.

Relational attributes relate the object to one or more other objects; i.e. describe the relationships among objects.

The machine repairman model includes the repairman, the machines, and the neutral location as objects. Each machine would have some unique identifier, perhaps a number, that is an indicative attribute. The repairman would have a location as a relational attribute to relate the repairman to a machine or the neutral location.

Indicative attributes can be classified as permanent or transitional.

A permanent indicative attribute can be assigned a value once, and only once, during model execution.

A transitional indicative attribute can be assigned a value more than once during model execution.

A status transitional indicative attribute can be assigned a value from a finite set of possible values.

A temporal transitional indicative attribute is assigned a value which is a function of time, i.e. the expression evaluated to assign a value to a transitional indicative attribute either contains sytem time or a temporal transitional indicative attribute.

The machine identifiers are examples of permanent indicative attributes. As an example of a status transitional indicative

attribute, each machine would also have an explicit or implicit attribute designating it as "failed" or "operating."[1] Examples of temporal transitional indicative attributes are the global clock for the model (called "system time" above) and the attribute giving the failure time of a machine.

> Relational attributes can be classified as hierarchical or coordinate.
>
> > An hierarchical relational attribute establishes a subordination of one object to another, implying that all characteristics of the subordinate object are descriptive of the superior object.
> >
> > A coordinate relational attribute establishes a bond or commonality between two objects based on: (1) the value(s) of one or more attributes, or (2) the appearance of one (or more) attribute(s) of one object in the expression for value assignment to one (or more) attributes of the other object.

The only hierarchical relationship in the machine repairman example is created possibly by the use of a "set," i.e. for example, an object representing all the machines. The set object is discussed more fully below. A coordinate relational attribute establishes the relation between the repairman location and a failed machine under repair, i.e. the repair of a machine requires the attribute for the repairman's loction to take on the value of the machine identifier.

---

[1] Implicit attributes are undefined descriptors whose values can be determined by the evaluation of expressions containing explicit attributes.

The relationships among objects, established by the relational attributes, can be used to characterize model structure. Of particular interest are the "bonds" between objects that seem distantly removed in a top-down model definition.

> A relational attribute is intermodular if it establishes a relation between two objects whose only common submodel is at the model level. Otherwise, a relational attribute is classified as intramodular.[2]

Relations among objects often are described by modelers through the use of "sets." We view the set relationships as specific instances of the more general classification presented above. For example, a coordinate attribute would describe the relation among set members; while the relation between the set object and any member would be described as hierarchical. However, the concept of "set" is extremely useful because of its descriptive capability.

> A set is an object that can be classified as either a primitive set (p-set) or defined-set (d-set).
>
> > A primitive set (p-set) is an object representing a collection of objects all of which have identically the same attributes.
> >
> > A defined set (d-set) is an object representing a collection of objects, not necessarily having the same attributes, defined by an expression evaluation during model execution.

---

[2] The degree of "bonding" can be described in more detail by an association measure, but the utility of this level of detail remains to be established.

The importance in distinguishing between p-sets and d-sets lies in the fact that members of p-sets can be characterized (completely described) and enumerated prior to model execution. The property of set membership is static. This static property does not hold for d-sets. Again, using the machine repairman example, the machines and their attributes can be recognized and such matters as storage allocation accomplished, at the initiation of model execution, but the number of machines failed at some point in time is known only then. In more complicated models, membership in d-sets, and the consequent effects, can be much more unpredictable. For example, a specification of "all vehicles with more than two axles, but not in tow, traveling in the left lane between mile markers 114 and 116" requires an expression evaluation involving several attributes, and the number of objects comprising the set might be difficult to estimate prior to the evaluation.

> A value can be typed (defined) as numeric, logical, or string.

The value types are so typical of those found in general purpose programming languages that their definition here is unnecessary. Relying again on the machine repairman example, each machine could have an attribute "operable" (a status transitional indicative attribute) which can be assigned the values "true" ("operational") or "false" ("failed"). A logical vaue ("true" or "false") would result from the evaluation of the expression: operable (machine 1) AND operable (machine 2). Note that more complicated descriptions can be created by resorting

to multi-valued logical attributes; for example, machine status: "failed under repair," "failed waiting repair," and "operational."

Every model must have an indexing attribute. While the indexing attribute commonly is "system time" or "time," the use of "space" is also plausible. Pertaining to the indexing attribute, we define terms that imply temporal relations linking the states of objects. Although a plausible argument can be made for "space" as an indexing attribute (with the consequence that spatial relations link the model states), we believe that the overwhelming usage of time justifies the reference to "system time" as the indexing attribute. Fundamentally, no limitation is imposed by this choice.

> An instant is a value of system time at which the value of at least one attribute of an object can be assigned (altered).

Keep in mind that a submodel or the entire model is an object, just as is a machine or the set of machines in the machine repairman example. Also, the definition of "instant" describes typical continuous simulation procedures, in which model states are updated on a fixed incremental basis (delta t) and system time is the single attribute whose value always changes.

> The state of an object is the enumeration of all attribute values of that object at a particular instant.
>
> (Consequently,) the state of a model is the exhaustive enumeration of the values of all attributes (of all objects) at a particular instant.

The state chronology (t1, t2) of an object is the enumeration of all attribute values of that object ordered by system time in the duration t1 to t2. (Note that t1 $\leq$ t2 and the duration includes all instants, rather than only those where an attribute of the object changes value.)

Durations of time are represented as intervals and spans.

An interval is the duration between two successive instants.

A span is the contiguous succession of one or more intervals.

The representations of time and state are coupled in the customary terms: event, activity, and process; but with the consequence that no ambiguity exists in the relation of one term to the others.

An activity is the state of an object over an interval.

An event is a change in object state, occurring at an instant, that initiates an activity precluded prior to that instant.

An event is <u>determined</u> if the only condition on event occurrence can be expressed strictly as a function of system time. Otherwise, the event is contingent.

An object activity is the state of an object between two events describing successive state changes for that object.

A process is the succession of states of an object over a span (or the contiguous succession of one or more object activities).

The benefits of the clarification of time and state relationships achieved with these definitions are explained in [NANCR81].

With respect to the machine repairman example, a machine (object) experiences an invariant pattern of behavior: an operational activity followed by an activity of failed awaiting repair which is succeeded by the activity of failed under repair, and this pattern repeats throughout simulation time. The process description of a machine focuses on the necessary conditions that permit one activity to terminate and its successor to begin. Note that the event of machine failure is determined (only a function of time), but the event of initiating repair is contingent on the location of the repairman at a failed machine. One further distinction remains among events.

> A simple event is a change in object state resulting from the change in value of a single attribute of an object.
>
> A compound event is comprised of two or more simple events.

This terminology, corresponding to the usage found in probability theory (for example, see [BHATU72, p. 333]), is useful in realizing a model representation that avoids the pathological issue of "zero time activity" (see [KIVIP73, p. 282-287]) and race conditions [PARND69].

III.3 Operations

Attributes of objects are the essential ingredients in simulation model representation. The categorization of attributes above is an attempt to describe the dynamic relations among objects.[3] Essentially, the value of an attribute is determined by an expression evaluation that includes the values of one or more other attributes combined with one or more operations. This section describes the necessary operations (and operators) for model representation.

III.3.1 Arithmetic and Logical Operations

The usual arithmetic and logical operations of a general purpose language are included: addition (+), subtraction (-), multiplication (*), division (/), conjunction (OR), disjunction (AND), and negation (NOT). The left arrow (<--) is the assignment operator. For the present we avoid the details of operator precedence and expression evaluation.

The reader must appreciate that at this level of detail we are treading a fine line between the recognition of necessary capabilities for applying the Conical Methodology and the stipulation of requirements for a Simulation Model Specification and Documentation Language (SMSDL)

---

[3] The description of dynamic relations using a static representation mechanism is a problem confounding several research areas in computer science.

that would achieve the implementation. Consequently, we view the listing of operations above as examples only and possibly incomplete. In all likelihood, a requirement of a SMSDL would be extensibility in both the operator set and data structures.

III.3.2 Object and Set Operations

The creation of an object requires the association (definition) of one or more attributes with that object, and the removal of all associations accomplishes the deletion of an object.

> Object creation is the definition of one or more attributes describing an object.
>
> Object deletion is the removal of all attributes defining an object.

Operations for set creation and set deletion establish or remove the relation(s) among set members.

> Set creation defines a set object ("header") having one or more relational attributes that provide access to the attribute values of all member objects.
>
> Set deletion removes the relationship established by at least one relational attribute.

The descriptive use of "sets" in the simulation context must be recognized as more specific than in the conventional mathematical usage,

where objects can be declared as set members without defining an explicit relationship. Simulation model representation requires that the relationship(s) be explicitly defined, but note that this requirement does not prevent the conceptual use of "set" without the explicit representation.

Further treatment of "sets" treads close to the domain of the SMSDL, which is beyond the scope of this paper. At this point we are content to consider a set as an object distinguishable from its member objects and characterized by one or more relational attributes and one or more value attributes.

III.4 Summary

The definitions above contribute to a structured, disciplined approach to model development. The term "Conical Methodology" is derived from the two stages in model development:

(1) Model definition through a "fan out" approach in identifying subsystems at increasingly lower levels until, given the input(s) and the objective(s), no further division into subsystems is warranted. This lowest object level is called the "base level."

(2) Model specification by the determination of value assignments to model attributes as objects are integrated into their defined submodels (which are objects) and submodels into higher level submodels (also objects) until the model level is reached.

The Conical Methodology is illustrated in the following section albeit without the benefit of a Simulation Model Specification and Documentation Language (SMSDL), which we have asserted to be the critical link between the modeler's conceptual abilities and model realization [NANCR77a, NANCR79] (see also [FRANE80]).

The definitions above also furnish the foundation for a more axiomatic approach to simulation model development. The three contrasting simulation "world views" — event scheduling, activity scan, and process interaction (following the Kiviat/Lackner categorization [KIVIP67, LACKM62]) — can be precisely defined. Further, we believe that analyses of model representations can contribute to the creation of less costly and more accurate models.

IV. An Example Application of the Conical Methodology

The machine repairman problem introduced in the previous section is used to demonstrate an application of the Conical Methodology. The outline format of the CM, modified from its original presentation

[NANCR77a, p. 16], is shown in Figure 4. Application of the CM is illustrated by the construction of a specific outline corresponding to Figure 4 for the machine repairman problem.

The reader is cautioned that the application without the SMSDL (Simulation Model Specification and Documentation Language) lacks the vehicle for smooth, relatively simple model creation. Furthermore, without the prompting and assistance of the interactive environment containing the SMSDL, the iteratively converging nature of the model development task cannot be fully appreciated.

I. Statement of Study Objectives

A. Definitions
B. Assumptions regarding objectives

II. Modeling Environment

A. Modeling effort

1. Organization creating model, dates, individuals, etc.
2. Scope of effort in time and money

B. Model assumptions

1. Boundaries
2. Interaction with environment

(a) Input description
(b) Assumptions on model/environment feedback or cross effects
(c) Output and format decisions

3. Initial state definition
4. Simulation termination conditions

III. Model Definition and Specification

A. Model

1. Sets
2. Indicative attributes
3. Relational attributes

B. Submodels

1. Submodel at the first level

(a) Sets
(b) Indicative attributes
(c) Relational attributes

((1)) Submodel at the second level

((a)) Sets
((b)) Indicative attributes
((c)) Relational attributes

.
.
.

(...(1)...) Object at level n

(...(a)...) Sets
(...(b)...) Indicative attributes
(...(c)...) Relational attributes

2. Submodel at the first level

.
.
.

IV. Model Validation and Verification Procedures

A. Validation tests
B. Verification criteria and tests

V. Model Experimentation

A. Hypotheses to be tested
B. Experimental design

VI. Implementation Requirements

Figure 4. The Conical Methodology Approach to Model Definition — Outline Illustration

IV.1 Model Definition

Completion of the outline of Figure 4 for the machine repairman problem follows below.

I. Statement of Study Objectives

To determine the system utilization for an assignment of a variable number of machines to a single repairman.

A. Definitions

N = number of machines

system utilization = the ratio of operational time to the total possible operational time for all machines

total machine downtime = sum of machine downtime (machine 1, ... ,machine N)

machine downtime = the wait time for repairman plus the machine repair time, for each machine failure

wait time for repairman = the time which the machine waits for repair, measured from its instant of failure to the beginning of repair on that machine

machine repair time = the time taken by the repairman to return a failed machine to the operational state

total repair time = sum of machine repair time, for each machine failure

B. Assumptions regarding objectives

1. No need to consider multiple repairmen — only the single repairman case is of interest.

2. The number of machines (N) could assume a value ranging from 2 to 50, specified by input.

3. A machine failure is immediately recognized by the repairman.

4. Individual machine downtimes should be recorded to permit possible consideration of differences among machines.

5. A total repair time is recorded in the event that operator utilization becomes of interest.

II. Modeling Environment

A. Modeling effort

1. Organization creating model, dates, etc.

Model created to illustrate the Conical Methodology by R. Nance on 8 August, 1979.

2. Scope of the effort in time and money

Model created in one staff-day and only to provide an example.

B. Model assumptions

1. Boundaries

Machines are arranged circularly so that the distances between adjacent machines are equal. Also, an equal distance separates the neutral location from any machine.

2. Interaction with environment

a. Input description

The number of machines (N) is specified as input as are: the distance between adjacent machines and the distance from any machine to the neutral location (both are given as time values — intransit time from/to the neutral location and intransit time from/to a machine).

The machine repair times follow a negative exponential distribution with the mean repair time (MU) specified by input. The machine failures occur according to a Poisson process with the mean time between failures (LAMBDA) specified as input.

The standard negative exponential distribution, designated by EXP, is provided so that only the single parameter specification is required.

b. Assumptions on model/environment feedback or cross effects

Only the states of "at location j" (a machine or the neutral location), travel to machines or travel to the neutral location ("intransit") are possible for the repairman. No rest breaks are represented.

Only the states of "operational," "failed awaiting repair," and "failed under repair" are possible for the machines. No dependency exists among machines (the failure events are independent).

No changes in input value specification during a run are permitted.

c. Output and format decisions

The values of the number of machines, mean repair time between failures (LAMBDA), and system utilization, with designating titles, are given as output.

3. Initial state definition

The initial state is defined as all machines in operation and the repairman at the neutral location. Thus, a time until failure must be assigned to all machines at system time set to zero.

4. Simulation termination conditions

For simplicity, a run is terminated when system time exceeds maxtime, a value prescribed by input.

III. Model Definition

A. Model

1. Sets

No sets are defined at the model level.

2. Indicative attributes

System time is a temporal transitional indicative attribute.
Units: minutes.

System utilization is a permanent indicative attribute, with value assignment at system time = maxtime.
Units: proportion (minutes/minutes).

Maxtime is a permanent indicative attribute, with value assignment at system time = zero.
Units: hours.

3. Relational attributes

None.

B. Submodels

1. Machine submodel

a. Sets

The machine submodel is based on the set of machines as a primitive set (p-set). The set "header" contains the number of machines (N) as an attribute.

The set of failed machines is a defined subset (d-set) of the set of machines. The set "header" contains the number of failed machines and the mechanism for referencing set members (as mentioned also under relational attributes).

b. Indicative attributes

Total machine downtime is a permanent indicative attribute.
Units: minutes.

The number of machines (N) is a permanent indicative attribute (that is contained in the set "header" of the set of machines).
Units: machines (integer with range 2 ... 50).

The time between failures for every machine follows a negative exponential distribution with mean LAMBDA, a permanent indicative attribute.
Units: minutes.

c. Relational attributes

A reference mechanism is necessary for the set of failed machines. (Note: this statement is implied by the definition of the set of failed

machines; every d-set must have a reference mechanism.)

(1) Base level of the machine submodel

Each machine is an object.

(a) Sets

Machines are members of the set of machines and can be members of the set of failed machines.

(b) Indicative attributes

Each machine has a machine identifier, a permanent indicative attribute, with a unique value.
Units: dimensionless.

Each machine has a machine clock and a machine downtime which are temporal transitional indicative attributes.
Units: minutes, for both.

Each machine has a condition attribute which is a status transitional indicative attribute.
Units: states, with range of "failed awaiting repair," "failed under repair," and "operational."

(c) Relational attributes

Each machine has membership in the set of failed machines based on the condition attribute value ("failed awaiting repair" or "failed under repair").

2. Repairman submodel

The repairman submodel can be defined immediately at the base level. The repairman is an object.

a. Sets

No sets are defined for the repairman object.

b. Indicative attributes

The machine repair time is a temporal transitional indicative attribute with value

generated from a negative exponential distribution with mean MU, a permanent indicative attribute which is identical for every machine failure.
Units: minutes for machine repair time and MU.

The repairman clock and the total repair time are temporal transitional indicative attributes.
Units: minutes, for both.

The intransit time from/to the neutral location to/from any machine is a permanent indicative attribute.
Units: minutes.

The repairman location and the intransit time between machines are status transitional indicative attributes.
Units: dimensionless for repairman location and minutes for intransit time between machines.

The machine intransit times make up a table of permanent indicative attributes.
Units: minutes.

c. Relational attributes

The repairman location is a relational attribute, relating the repairman (object) to the machine objects.

At this point the first stage — model definition — is terminated (but not necessarily completed since model development can iterate through more than one definitional phase). Model specification in the second stage is guided by the model definition, which is summarized for convenience in Figure 5.

A. Model

1. Sets: None

2. Indicative attributes:
System time: temporal transitional indicative attribute
System utilization: permanent indicative attribute
Maxtime: permanent indicative attribute

3. Relational attributes: None

B. Submodels:

1. Machine submodel:

a. Sets: Set of machines (p-set) with number of machines
Set of failed machines (d-set) with number of failed machines

b. Indicative attributes:
Total machine downtime: permanent indicative attribute
Number of machines: permanent indicative attribute
Number of failed machines: status transitional indicative attribute
Mean failure rate (LAMBDA): permanent indicative attribute

c. Relational attributes: Reference mechanism for set of machines and set of failed machines

(1) Base level of machine submodel: Each machine is an object

(a) Sets: Members of the set of machines and can be members of the set of failed machines

(b) Indicative attributes:
Machine identifier: permanent indicative attribute
Machine clock: temporal transitional indicative attribute
Machine downtime: temporal transitional indicative attribute
Wait time for repairman: temporal transitional indicative attribute
Condition: status transitional indicative attribute
Time between failures: temporal transitional indicative attribute

(c) Relational attributes: A coordinate relation exists among members of the set of failed macines identified by the value of the condition attribute

2. Repairman submodel (Base level)

The repairman is an object.

a. Sets: None

b. Indicative attributes:
Total repair time: temporal transitional indicative attribute
Machine repair time: temporal transitional indicative attribute
Mean repair rate (MU): permanent indicative attribute
Repairman clock: temporal transitional indicative attribute
Repairman location: status transitional indicative attribute
Intransit time to/from neutral location: permanent indicative attribute
Intransit time from a machine: status transitional attribute
Machine intransit time table: permanent indicative attributes

c. Relational attributes: Repairman location relates the repairman (object) to the machine objects.

Figure 5. Summary of the Model Definition Stage (Top-Down)

IV.2 Model Specification

Model definition relies heavily on the modeler's observational capability: the top-down description is a statement of what exists complicated only by the need to classify attributes. The model specification requires a more indepth recognition of the interactions among attributes, particularly as these interactions vary with time. During this stage the SMSDL should exert a crucial influence through its ability to promote an iterative convergence in the derivation of the desired specification. We can give only an inadequate approximation of this derivation which, although to the modeler should seem free and unrestrictive, is carefully structured.

Model specification can begin at several points, no one clearly more appropriate than another. Any base level submodel is a candidate. Actually, any attribute with value assigned by input can be immediately specified. One possibility is to select that base level submodel with the maximum number of attributes; yet, an argument could be made to the contrary.[4]

Somewhat arbitrarily we begin with the machine submodel since machine failures are the "driving events" in the model. Initially, all machines are in the "operational" state, and the repairman is at the neutral location. A change in state occurs with the first machine to experience a failure, and at this instant the repairman also changes

---

[4] Ordering of the base level submodel specifications can be made more specific; the criteria are the subject of ongoing research.

state (from "at neutral location" to "intransit"). Repair cannot begin until the repairman is at the failed machine. In general, the period until repair begins is comprised of both intransit time and the time for repair of machines failed and not repaired prior to the failure of the machine under consideration. (Of course, the first failed machine has no predecessors, but the general case is of more interest.)

In order to clarify the assignment of values to attributes, we use the temporal subscripts:

t_f = future time (a value > system time),

t_o = current value of system time, and

t_p = past time (a value < system time).

Also, the set of failed machines is represented by F and the particular failed machine under repair is designated as having j as the value of the machine identifier.

At the instant of the repairman's arrival at the failed machine, we begin the update of attribute values.

$$j \text{ wait time for repairman}(t_o) \leftarrow \text{system time } (t_o) - \underset{\text{clock}(t_p)}{j \text{ machine}}$$

$$j \text{ condition}(t_o) \leftarrow \text{"failed under repair"}$$

At this point we need to treat the repairman submodel attributes which describe the interaction of the repairman with the failed machine. Specifically,

machine repair time(t_o) <-- EXP(MU),

total repair time(t_o) <-- total repair time(t_p) + j wait time for repairman(t_o), and

system time(t_f) <-- system time(t_o) + machine repair time(t_o).

With an updated value of system time, i.e. the current value of t_o is the previously updated system time(t_f), we continue:

repairman clock(t_o) <-- system time(t_o), and

j machine downtime(t_o) <-- machine repair time(t_p) + j wait time for repairman(t_p).

Updating the machine clock to mark the next failure of the j machine is accomplished by utilizing the EXP function.

time between failures(t_o) <-- EXP(LAMBDA),

j condition(t_o) <-- "operational", and

j machine clock(t_f) <-- system time(t_o) + time between failures(t_o)

At this instant the repair is completed (with the consequent update of the clock for the machine just repaired) and the next state of

the repairman is "intransit," but to what location? The selection of the next location is the crux of the major event in the model. If at least one machine is failed, i.e. the set of failed machines is not null, then the repairman moves to the location j, the "first" of the failed machines. After updating the repairman clock, we resume with the sequence of computations described above. However, if no machine is failed, the repairman moves to the neutral location. The next location is determined by the functional representation shown below.

$$\text{repairman location}(t_f) \longleftarrow \begin{cases} \text{k machine first in F if } |F| > 0 \\ \\ \text{neutral location, otherwise.} \end{cases}$$

The first condition above leads to an intransit time between machines specified by a table access with j (the current repairman location) and k (the next machine location) as arguments. The update of system time to reflect the intransit time is accomplished by the update of the repairman clock (note that the determination of a future value of system time always redefines the current time).

$$\text{intransit time between machines } j,k(t_o) \longleftarrow \text{machine intransit time table } \langle j,k \rangle$$

$$\text{system time}(t_f) \longleftarrow \text{system time}(t_o) + \text{intransit time between machines } j,k(t_o)$$

$$\text{repairman clock}(t_o) \longleftarrow \text{system time}(t_o)$$

The second condition, which leads to the state "at neutral location" for the repairman, also introduces the possibility of time passing while the repairman waits for a failure. To capture this situation we update the value of system time to place the repairman at the neutral location (and update the repairman clock) then follow with a second update depending on the status of the machines at that instant.

$$\text{system time}(t_f) \leftarrow \text{system time}(t_o) + \text{intransit time from/to neutral location}$$

$$\text{repairman clock}(t_o) \leftarrow \text{system time}(t_o)$$

$$\text{system time}(t_f) \leftarrow \begin{cases} \text{system time}(t_o) + \text{intransit time from/to neutral location if } |F| > 0 \\ \min(\text{k machine clock}(t_f)) + \text{intransit time from/to neutral location, otherwise.} \end{cases}$$

The selection of the minimum machine clock identifies the location of the next failure, designating the repairman location (at the current value of system time).

$$\text{repairman location}(t_o) \leftarrow k \mid k \text{ machine clock}(t_o) \text{ is minimum.}$$

The "core" of the model is now completed, since we have returned to the state of the repairman at the designated failed machine to begin repair. Summary attributes, such as total machine downtime and system utilization, and the test for termination by comparing the value of system time with maxtime remain to be specified. The complete

specification is shown in the series of figures relating the model definition and specification stages (Figures 6 - 9). We note again that the modeler would be expected to cycle between the definition and specification stages several times during the development of a model, rather than simply proceeding through a top-down definition succeeded by a bottom-up specification as described above.

Definition Stage		Specification Stage
Component or Attribute	Component or Attribute Type	Specification Assignment [Number of Occurrences]
(a) Sets		
set of machines--members	coordinate (p-set)	(next machine) initialization
set of failed machines--members	coordinate (d-set)	(next failed) j condition = "failed under repair" V "failed awaiting repair"
(b) Indicative attributes		
machine identifier	permanent	input
machine clock	temporal transitional	j machine clock(t_f) <— system time(t_o) + time between failures(t_o)
machine downtime	temporal transitional	j machine downtime(t_o) <— machine repair time(t_p) + j wait time for repairman(t_p)
wait time for repairman	temporal transitional	j wait time for repairman(t_o) <— system time(t_o) - j machine clock(t_p)
condition	status transitional	j condition <— "failed under repair" [2] j condition <— "operational"
time between failures	temporal transitional	time between failures(t_o) <— EXP (LAMBDA)
(c) Relational attributes		
next machine	intramodular	j next machine <— { j + 1 for j = 1 ... number of machines - 1 0 for j = number of machines
next failed	intramodular	

Figure 6. Specification of the Machine Base Level

Definition Stage		Specification Stage
Component or Attribute	Component or Attribute Type	Specification Assignment [Number of Occurrences]
(a) Sets. None		
(b) Indicative attributes		
total repair time	temporal transitional	total repair time(t_o) <-- total repair time(t_p) + machine repair time(t_o)
machine repair time	temporal transitional	machine repair time(t_o) <-- EXP(MU)
MU	permanent	input
repairman clock	temporal transitional	repairman clock(t_o) <-- system time(t_o) [3]
repairman location	status transitional	repairman location(t_f) <-- { k machine first in |F| if |F| > 0; neutral location, otherwise
		repairman location(t_o) <-- k|k machine clock(t_o) is minimum [2]
intransit time to/from neutral location	permanent	input
intransit time between machines	status transitional	intransit time between machines j,k(t_o) <-- machine intransit time table <j,k>
machine intransit time table	permanent	input
(c) Relational attributes		
repairman location	intermodular	specified above

Figure 7. Specification of the Repairman Submodel (Base)

Definition Stage		Specification Stage
Component or Attribute	Component or Attribute Type	Specification Assignment [Number of Occurrences]
(a) Sets		
set of machines--header	hierarchical (p-set)	(first machine) initialization
set of failed machines--header (F)	hierarchical (d-set)	(first failed) <-- { j | j condition = "failed" : j machine clock(t_p) is minimal ; 0 : otherwise
(b) Indicative attributes		
number of machines (N)	permanent	number of machines <-- input
total machine downtime	permanent	total machine downtime (maxtime) <-- for j:1...number of machines (Sum (j machine downtime(t_o))
number of failed machines	status transitional	number of failed machines <-- { number of failed machines + 1 for j machine clock(t_o) $\leq$ system time $\wedge$ j $\notin$ |F| ; number of failed machines - 1 for j machine clock(t_o) > system time $\wedge$ j $\in$ |F|
LAMBDA	permanent	input
(c) Relational attributes		
first machine	intramodular	
first failed	intramodular	

Figure 8. Specification of the Machine Submodel

Definition Stage		Specification Stage
Component or Attribute	Component or Attribute Type	Specification Assignment [Number of Occurrences]
(a) Sets. None		
(b) Indicative attributes		
system time	temporal transitional	system time(t_f) <-- system time(t_o) + machine repair time(t_o) system time(t_f) <-- system time(t_o) + intransit time between machines j,k(t_o) system time(t_f) <-- system time(t_o) + intransit time to/from neutral location [4] system time(t_f) <-- { system time(t_o) + intransit time from neutral location if |F| > 0; min {k machine clock(t_f)} + intransit time to/from neutral location; otherwise
system utilization	permanent	system utilization <-- 1 - total machine downtime/(number of machines * maxtime)
maxtime	permanent	input
(c) Relational attributes. None		

Figure 9. Specification of the Model Level

As a final comment we return to the two sets — the set of machines and the set of failed machines. The former is a p-set, with all of its characterizing attributes established prior to model execution. The latter, a d-set, is made up of all machines having the value "failed awaiting repair" plus the possible single machine with value "failed under repair" for the condition attribute. While the definition of the set of failed machines adds little to the model representation in this case (the selection of the machine to be repaired could be made strictly on a comparison of machine clock values), we have included it to illustrate the use of sets, which proves exceedingly useful with more complicated models.

IV.3 Summary

The machine interference problem has served to illustrate an application of the Conical Methodology for model representation. Completion of the model development would include validation and verification, experimentation, and implementation, omitted here in the interest of brevity. In the following section we examine both the development process and the produced representation to critique both the process and the product.

V. The Methodology and the Representation: An Incomplete Critique

The illustration of the Conical Methodology in Section IV only confirms that one model can be developed according to a top-down definition and bottom-up specification based on the attribute classifications of Section III. What is gained by the sacrifice of time and thought to produce this structured, categorized representation summarized in Figures 6 - 9? That question is the subject of this section, but no definitive answer is possible at this time for the SMSDL remains a crucial missing component in model development. Nevertheless, the application of the Conical Methodology and the product itself impart further knowledge about the characteristics of an SMSDL and simulation model development. We attempt to organize this knowledge in the second subsection.

V.1 Examination of the Model Representation

Observations on the machine repair model are presented as brief comments, in both a positive and negative vein and sometimes combined. The reader is advised that references to Section IV, and perhaps to Section III as well, are helpful in evaluating the comments.

V.1.1 Considerable redundancy exists in the representation.

Redundancy is present in several forms: between the definition of sets and the definition of relational attributes for example, or the use of a condition attribute for machines to indicate a failed state when a comparison of the machine clock with system time provides the same information. However, redundancy is not undesirable in the development of a model, for it provides a means of checking for consistency (see [ZEIGB79, p. 108]) and assists in assuring completeness.

Redundancy in the model development should not be confused with redundancy in the experimental model, represented in a SPL. The latter is a potential source of inefficiencies in execution time and/or simplification of the experimental representation. For example, the explicit use of a repairman clock is revealed to be unnecessary since its specification is always the assignment of the value of system time; that is, the value of the repairman clock is simply obtained by assigning the value of system time.

V.1.2 The hierarchical development in Figure 4 is too restrictive.

The outline of Figure 4 serves to illustrate the top-down definitional decomposition, which is followed by the bottom-up composition in the specification stage. This format does not illustrate

the procedure by which the model is developed. Model development is an iterative procedure, providing for interchanges between the definition and specification stages so as to converge on the model representation through successive refinements.

V.1.3 The modeler should not have to prescribe the reference mechanisms for sets or, in general, define and specify at this level of detail.

We agree that this example has probably carried the representation into the domain of the SPL. We have erred in this direction intentionally so as to emphasize the descriptive power one can use. A SMSDL must permit the level of description to range from very high to very low levels, according to the choice and needs of the modeler as we have noted elsewhere [NANCR79, p. 93]

V.1.4 The representation provides a basis for testing consistency in the use of attributes.

In addition to the detection of inconsistencies through redundant description, the classification of attributes enables the identification of differences between definition and specification. For example, having classified machine downtime as a temporal transitional indicative attribute, the modeler would be alerted to his specification

of the attribute as an expression devoid of attributes which are functions of time (or a generated function value such as EXP). By including dimensionality requirements, the detection of inconsistencies is made more exacting through the ability to match the dimensionality of an assignment expression with that of the assigned attribute.

V.1.5 Model completeness is facilitated through the ability to check on attributes defined but not used in the specification and vice versa.

Completeness is an important property to assess of any model. While the Conical Methodlogy cannot assure completeness with certainty, the use of an undefined attribute in the specification of another attribute can be detected easily. Likewise, the definition of an attribute that never appears in the specification expression of another attribute is recognizable.

V.1.6 The representation provides a basis for the estimation of relative model complexity.

At this juncture such an assertion is based primarily on faith or hope. However, this optimistic claim has some foundation. Currently, program complexity is recognized as an exceedingly important area that has attracted much attention without admitting much subsequent understanding. The analysis of structural relationships through

directed graph models of data or control flow have furnished only limited insights into the inherent complexity of programs. By classifying attributes, we construct a more substantial relationship for investigation.

The complexity of a simulation model seems intimately related to the degree of interaction among objects in the model. This "degree of interaction" could be indicated in several ways. Consider the following possible indicators of model complexity:

(1) the number of statements required to update system time,

(2) the number and types of attributes included in the expressions for determining the updated values of system time,

(3) the number of intermodular relational attributes,

(4) the mix of attribute types in the expressions for value assignments to temporal transitional indicative attributes, and

(5) the directed graph showing the relationships among all attributes (which is related to the component interaction diagram of Zeigler [ZEIGB76, p. 21]).

Any or all of these might prove necessary to comprise an adequate and useful measure of model complexity.

V.1.7 The representation provides very good documentation support without additional effort.

The model documentation is enabled as a byproduct of the definition and specification activities. If desired, each level of the conical partitioning can be described without recourse to further detail. The model and first submodel levels might prove sufficient to explain the model to a manager; while a programmer considering the required data structures might access the base level descriptions. Relationships among attributes are clearly depicted in the specification stage; thus permitting the "domino effect" of changes in object descriptors to be perceived a priori. Further, sensitivity tests can be conducted more selectively since the first level effects of changes in parameters and structures can be recognized.

V.2 The Conical Methodology for Simulation Model Development

V.2.1 The model development activity is too creative a task to be so tightly structured.

The absence of a SMSDL and the corequisite environment forces the development task to appear as sequential and narrowly channeled. Such is neither mandated nor intended. A modeler need not proceed in a strict top-down fashion through model definition followed by the bottom-up specification. On the contrary, one can define submodels with different superior submodels, and can interchange between the definition and specification stages easily. The Conical Methodology also serves as an overseer — monitoring the relationships among submodels, revealing submodel incompleteness, and detecting certain inconsistencies in object descriptors.

V.2.2 The Conical Methodology and the requisite SMSDL will be costly systems.

The development of the SMSDL is a costly, time-consuming effort. Similarly, the cost of developing and implementing the requisite environment for application of the Conical Methodology — a Model Management System — is estimated to cost much more than the SMSDL alone. However, in comparison with the cost of developing a single

simulation model similar to those for the validation of defense systems for example, the cost is relatively low. The operational costs will not prove burdensome, and will reduce the cost of developing large simulation models. An interactive environment is a necessity so that: (1) the model development system can prompt the modeler, (2) the modeler can effect construction and revision in concert with the system, (3) the interchange between definition and specification can be rapid to preserve the recognition of associated structures and consequent implications, and (4) partial models and submodels can be stored and retrieved over the duration of a lengthy simulation study, permitting reflective periods and providing convenient communications among members of a modeling team.

V.2.3 The Conical Methodology is simply the application of the program development techniques, often described as "structured programming," to simulation modeling.

Elements of the philosophies and approaches of several program development techniques are recognizable in the CM just as these techniques can be characterized as applications of the engineering approach (subdivide, analyze, and synthesize), and, in turn, the engineering approach constitutes an extension of the scientific method, which finds its roots among the early mathematicians and philosophers.

To the extent that a program or system development task is viewed as a modeling activity, that task shares the same objectives and assumes a resemblance to simulation model development. However, we maintain that certain characteristics of the Conical Methodology are not so apparent in the popular program development techniques. For example, the Conical Methodology:

(1) seeks to support the entire simulation experiment rather than just the model development,

(2) incorporates diagnostic evaluation of the modeling results so as to provide corrective improvements during the model development,

(3) treats the production of multilevel documentation as an integral part of model development, and

(4) includes objectives that, to be accomplished, require a closer interaction with the modeler than might be necessary with a program development technique.

V.2.4 The Conical Methodology provides an underlying structure that accommodates a more axiomatic approach to simulation model development and experimentation.

The top-down definition and bottom-up specification furnish an explicit template for model development that assists in the assurance of completeness. Attribute classification enables certain checks for consistency. Taken together, the two provide a potential for analyzing relative model complexity, which can contribute to a priori characterization of the modeling effort and to more accurate estimation of the required resources, time, and documentation effort.

V.2.5 The methodology offers a model representation and a development environment appropriate to the realization of knowledge-based experimentation so crucial to large simulation studies.

The necessity for continued assistance in simulation experimentation during the originating study, a subsequent modification, a follow-on extension, or a reapplication to a different problem area has been identified as important by Mathewson [MATHS78, MATHS79] and by Oren and Zeigler [ORENT79]. Specific capabilities have been proposed and, in a few cases, implemented by Maroglio [MAROL79] to support large simulation models for a U.S. Navy application. The Conical Methodology contributes a basis for realization of a knowledge base to guide model experimentation and provide a vehicle for retrospective inquiry into

test cases and results. Outcomes of individual experiments can be integrated in ways to address experimental questions unforeseen initially and to extract behavioral responses requiring the linking of experiments.

V.3 Summary

This examination of the model representation and the representation development process has been labeled "incomplete" because of the absence of a SMSDL. Both a SMSDL and the environment created by a Model Management System are crucial to the Conical Methodology. Nevertheless, the assertions posed in this section and the responsive comments help the reader to understand our perceptions and motivations and to obtain answers to some of the more apparent questions. A conclusive assessment of the Conical Methodology must remain as a future responsibility.

References

BHATU72 Bhat, U. Narayan. Elements of Applied Stochastic Processes, John Wiley, 1972.

BLUNG67 Blunden, G.P. and H.S. Krasnow. "The Process Concept as a Basis for Simulation Modeling," Simulation, 9(2): August 1967, 89-93.

CLEMA73 Clementson, Alan T. "Extended Control and Simulation Language," University of Birmingham, Birmingham, England, 1973.

CONTA77 Control Analysis Corporation, "Report on Feasibility of and Methodology for Developing Guidelines for Large-Scale Models," Report to the National Bureau of Standards, January 24, 1977.

COXDR62 Cox, D.R. and Walter L. Smith. Queues, Methuen & Co. Ltd., 1961.

CROOJ80 Crookes, John G. (personal communication) August 22, 1980.

DAVIN76 Davies, N.R. "A Modular Interactive System for Discrete Event Simulation Modeling," Proceedings of the Ninth Hawaii International Conference on System Sciences, January 10, 1973.

DECAR76 DeCarvalho, R. Spinelli and John G. Crookes. "Cellular Simulation," Operational Research Quarterly, 27(1): 1976, 31-40.

ELZAM79 Elzas, M.S. "What Is Needed for Robust Simulation?" in Methodology in Systems Modeling and Simulation, B.P. Zeigler, T.I. Oren, M.S. Elzas, and G.J. Klir (eds.), North Holland: Amsterdam, 1979, pp. 57-91.

ETSCM72 Etschmaier, M.M. "Mathematical Modelling for the Simulation of Discrete Systems," Technical Report No. 3, Department of Industrial Engineering, University of Pittsburgh, June 1972.

FRANE80 Frankowski, Elaine N. and W.R. Franta, "A Process Oriented Simulation Model Specification and Documentation Language," Software - Practice and Experience, 10: 1980, 721-742.

HEIDG74 Heidorn, George E. "English as a Very High Level Language for Simulation Programming," SIGPLAN Notices (Proceedings of the Symposium on Very High Level Languages), 9(4): April 1974, 91-100.

HILLR69 Hills, P.R. and T.G. Poole "A Method for Simplifying the Production of Computer Simulation Models," TIMS Tenth American Meeting, Atlanta, October 1-3, 1969.

HOLBA77 - Holbaek-Hanssen, E., P. Handlykken, and K. Nygaard. "Systems Description and the DELTA Language, Report No. 4 (Publication No. 523), second printing, Norwegian Computing Center, Oslo, 1977.

KINDE76 Kindler, Evzen "On the Way to a Mathematical Theory of Simulation," Elektronische Informationsverarbeitung und Kybernetik 12: 1976, 497-504.

KINDE77 Kindler, Evzen "Mathematical Theory of Static Systems," Kybernetika, 13 (3):1977, 176-189.

KINDE78 Kindler, Evzen "Classification of Simulation Programming Languages: I. Decelaration of Necessary System Conceptions, "Elektronische Informationsverarbeitung und Kybernetik 14 (10):1978, 519-526.

KINDE79 Kindler, Evzen "Dynamic Systems and Theory of Simulation," Kybernetika, 15 (2):1979, 77-87.

KIVIP67 Kiviat, Philip J. "Digital Computer Simulation: Modeling Concepts," Rand Corporation Memorandum RM-5378-PR, Santa Monica, California, 1967.

KIVIP73 Kiviat, P.J., R. Villaneuva, and H.M. Markowitz. SIMSCRIPT II Programming Language, CACI Inc., 1973.

KLEIH77a Kleine, Henry. "SDDL: Software Design and Documentation Language," Publication 77-24, Jet Propulsion Laboratory, Pasadena, California, July 1, 1977.

KLEIH77b Kleine, Henry. "A Vehicle for Developing Standards for Simulation Programming," Proceedings 1977 Winter Simulation Conference, Gaithersburg, Maryland, December 5-7, 1977, 730-741.

LACKM62 Lackner, M.R. "Toward a General Simulation Capability," Proceedings of the SJCC, AFIPS Press, May 1962, pp 1-14.

LACKM64a Lackner, M.R. and P. Kribs. "Introduction to the Calculus of Change," Systems Development Corporation TM-1750/000/01, February 12, 1964.

LACKM64b Lackner, M.R. "Digital Simulation and System Theory," Systems Development Corporation SP-1612, April 6, 1964.

LAFFH73 Lafferty, H.H. "Sheffield Simplified Simulation System - S4 Manual," University of Sheffield, Department of Applied Mathematics and Computing Science, January 10, 1973.

LEHMM80 Lehman, M.M. "Programs, Programming and the Software Life Cycle, "Report No. 80/6, Department of Computing and Control, Imperial College of Science and Technology, London, April 15, 1980.

MARKH77 Markowitz, Harry M. "SIMSCRIPT," Research Report RC 6811(#29158), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 13, 1977.

MARKH79 Markowitz, Harry M. "Proposals for the Standardization of Status Description," Research Report RC 7782(#33671), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, July 26, 1979.

MAROL79 Maroglio, Louis J. (personal communication) November 1979.

MATHS74 Mathewson, S.C. "Simulation Program Generators," Simulation, 23 (6): December 1974, 181-189.

MATHS75 Mathewson, S.C. "Interactive Simulation Program Generators," Proceedings of The European Computing Conference on Interactive Systems, Brunel University, 1975.

MATHS78 Mathewson, S.C. "Computer Aided Simulation Modeling and Experimentation," Proceedings of the Eighth Australian Computer Conference, 1978, pp. 9-13.

MATHS79 Mathewson, S.C. "Integrated Computer Simulation Modeling," (privately communicated), 1979.

MATHS80 Mathewson, S.C. "User Acceptance: Design Considerations for a Program Generator," Department of Management Science, Imperial College, September 1980.

MCLEJ70 McLeod, J. "Toward Uniform Documentation — PHYSBE and CSMP," Simulation, 14(5): May 1970, 215-220.

MCLEJ73 McLeod, J. "Simulation: from Art to Science for Society," Simulation, 21(6): December 1973 (SIMULATION TODAY, Center Section).

NANCR72 Nance, Richard E. "Towards a Unifying Structure for Discrete Event Simulation," SIMULETTER, 3(2): January 1972, 4-9.

NANCR77a Nance, Richard E. and Anil Chatterji, "The Feasibility of and Methodology for Developing Federal Documentation Standards for Simulation Models," Interim Report to the National Bureau of Standards, Department of Computer Science, Virginia Tech, April 11, 1977.

NANCR77b Nance, Richard E. "The Feasibility of and Methodology for Developing Federal Documentation Standards for Simulation Models," Final Report to the National Bureau of Standards, Department of Computer Science, Virginia Tech, June 26, 1977.

NANCR79 Nance, Richard E. "Model Representation in Discrete Event Simulation: Prospects for Developing Documentation Standards," in Current Issues in Computer Simulation, N. Adam and A. Dogramaci (eds.), Academic Press, 1979, pp. 83-97.

NANCR81 Nance, Richard E. "The Time and State Relationships in Simulation Modeling," Communications of ACM, 24(4): April 1981, 173-179.

OLDFP66 Oldfather, P.M., A.S. Ginsberg, and H.M. Markowitz. "Programming by Questionnaire: How to Construct a Program Generator," RAND Report RM-5129-PR, November 1966.

OLDFP67 Oldfather, P., A.S. Ginsberg, P.L. Love, and H.M. Markowitz. "Programming by Questionnaire: The Job Shop Simulation Program Generator," RAND Report RM-5162-PR, July 1967.

ORENT78 Oren, Tuncer I. "A Personal View on the Future of Simulation Languages," Proceedings of the UKSC 1978 Conference on Computer Simulation, Chester, April 1978, pp. 294-306.

ORENT79 Oren, Tuncer I. and Bernard P. Zeigler, "Concepts for Advanced Simulation Methodologies," Simulation, 32(3): March 1979, pp. 69-82.

ORENT80 Oren, Tuncer I. "Computer-Aided Modeling Systems (CAMS)," Plenary Address, Simulation '80 Symposium, Interlaken, Switzerland, June 25-27, 1980.

PALMD47 Palm, Docent C. "The Distribution of Repairman in Servicing Automatic Machines," (Swedish), Industritidningen Norden, Volume 175, p. 75.

PARND69 Parnas, David L. "On Simulating Networks of Parallel Processes in which Simultaneous Events May Occur," Communications of ACM, 12(9): September 1969, 519-531.

POOLT77 Poole, T.G. and Jan Szymankiewicz. Using Simulation to Solve Problems, McGraw-Hill, 1977.

RICHM80 Richerson, Michael R. "Some Techniques Found Useful in the Development of a Large-Scale Simulation Model," Simulation, 34 (5): May 1980, 177-178.

ROTHP78 Roth, Paul F., Saul I. Gass and Austin J. Lemoine. "Some Considerations for Improving Federal Modeling," Proceedings of the Winter Simulation Conference, December 3-5, 1978.

SCICS67 "The SCi Continuous System Simulation Language (CSSL)," Simulation, 9(6): December 1967, 281-303.

SHANR70 Shannon, R.E. and W.E. Biles. "The Utility of Curiculum Topics to Operations Research Practitioners," Operations Research, 18 (4): July - August 1970, 741-745.

SHANR80 Shannon, Robert E., S. Scott Long and Billy P. Buckles. "Operations Research Methodologies in Industrial Engineering," AIIE Transactions, 12(4): December 1980, 364-367.

SOLOS80 Solomon, Susan. "Building Modelers: Teaching the Art of Simulation," INTERFACES, 10(2): April 1980, 65-72.

TOCHK64 Tocher, K.D.T. "Some Techniques of Model Building," Proceedings IBM Scientific Computing Symposium on Simulation Models and Planning, White Plains, New York, December 7-9, 1964, 119-155.

USGAO76 U.S. Government Accounting Office, "Ways to Improve Management of Federally Funded Computerized Models," LCD-75-111, Washington, D.C., August 23, 1976.

VIDAC80 Vidallon, C. "GASSNOL: A Computer Subsystem for the Generation of Network Oriented Languages with Syntax and Semantic Analysis," Simulation '80 (preprint), Interlaken, Switzerland, June 25-27, 1980.

VISOG79 Visontay, Gy. and P. Csaki. "DOCUM — for Automatic Documentation of SIMULA Programs," SIMULA Newsletter, 7 (2): May 1979.

ZEIGB76 Zeigler, Bernard P. Theory of Modeling and Simulation, John Wiley & Sons: New York, 1976.