Simulation Model Development: System Specification Techniques

Lynne F. Barger
Richard E. Nance

TR 86-19

Technical Report TR-86-19†

SIMULATION MODEL DEVELOPMENT: SYSTEM SPECIFICATION TECHNIQUES*

Lynne F. Barger
Richard E. Nance

Department of Computer Science
and
Systems Research Center
Virginia Tech
Blacksburg, Virginia 24061

* This research supported by the U.S. Navy through the Systems Research Center under the Basic Ordering Agreement N60921-83-G-A165 B001-Z

† Cross-referenced as Technical Report SRC-86-005, Systems Research Center, Virginia Tech.

ABSTRACT

A review of software specification techniques and specification languages is described. Special emphasis is given to simulation model specification and the degree to which general software techniques are applicable in the modeling domain. The role of specification is examined in terms of existing techniques as well as abstraction permitting the identification of desirable properties unrelated to existing tools. A categorization of specification languages assists in understanding the similarities and differences among current approaches. Important conclusions are that: (1) specification methodologies are highly dependent on the phase of the software life cycle, (2) both the general software and simulation communities lack a clear perception of the role of specification, and (3) tools and environments tend to lack an underlying methodological foundation.

CR Categories and Subject Descriptors: D.3.1 [**Programming Languages**] Formal definitions and Theory; I.6.2 [**Simulation and Modeling**] Simulation Languages

Keywords: simulation model specification, model documentation, language categories, model development environments

1. SIMULATION MODEL DEVELOPMENT

When simulation programming languages (SPLs) first appeared, little guidance in **HOW** to develop a simulation model was available. The only sources of assistance were books and manuals describing the simulation programming languages. With the development of automatic program generators, the modeler received slightly more guidance, but the target SPL remains as a constraining force on the model representation produced by the generator. Currently, developing a general theory of simulation, which is independent of any SPL, is viewed as one path toward extending the capabilities for dealing with large, complex discrete event simulation models. This report describes the several components that could prove important in charting this path.

A motivation is the research reported herein is the investigation of the comparative roles of *specification* in two domains: discrete event simulation and embedded systems software development. While the effort does not provide conclusive findings, some preliminary indications are perceived, and several conclusions are stated in the final section.

1.1 Simulation Programming Languages

A simulation programming language offers a conceptual framework for model development which is based on the world view of the language. The advantage of using a particular world view is that one begins with a preconceived notion of how to decompose a model into its essential parts.

Three world views have been identified [KIVIP69], and each world view produces a very different model representation of the same model [OVERM82, ZEIGB84b]. The event world view emphasizes a decomposition based on when actions occur in the model (time emphasis).

The activity world view yields a decomposition based on conditions in the model (state emphasis). The process world view stresses what happens to a particular object in the model (object emphasis). Thus, the modeler can choose to decompose based upon time, state, or object, and the choice reflects a *locality* of descriptive emphasis [OVERM86].

The use of a SPL for model development is not without problems. Since a SPL is a programming language, it contains many features that are fundamental in programming a simulation (executing the model) that are unnecessary, perhaps even inhibitive, during model decomposition and description. The modeler is unable to separate these features, and thus becomes concerned with programming details too early in the development process [BALCO86a]. Also the diverse views taken by the various SPLs make it difficult for the modeler to conceptualize any basic principles of model development. This problem is compounded by each world view using a different definition of the terms **event, activity,** and **process** [NANCR81b]. A more serious problem arises when the modeler becomes too accustomed to one world view embedded in a particular SPL and slavishly decomposes every model according to that SPLs conceptual framework [OVERM82]. The modeler may overlook important details because those details do not conform to the selected world view. Even important conceptual characteristics may be subjugated by the inability of the world view to provide a natural representation [HENRJ83]. The resulting model may prove biased and have little direct correspondence to the actual system being modeled. Even if the model is valid, the imposed conceptual framework could prove costly in the experimentation and maintenance phases of the model life cycle [BALCO86a].

In conclusion, an SPL is lacking in the support of model development due to the lack of generality [SUBRE81], the overemphasis on programming details, and a general lack of expressive power for describing system characteristics that tend to be incongruous with the embedded world view.

1.2 Program Generators

A program generator takes as input a model description in a specific format and automatically generates syntactically correct code in some simulation programming language [NANCR81a, ZEIGB84b, DAVIN79]. The model description is often obtained interactively via questionnaire. The questionnaire solicits from the modeler relevant model properties without concerning the modeler about implementation details.

The goal of automatic code generation limits the class of models that can be addressed. These models must have well defined structural properties [ZEIGB84b, DAVIN79, MATHS84], and the properties stressed are influenced by the world view of the target language. Models successfully treated by the application of program generators are typically simple queueing models [DAVIN79, SUBRE81] and related models that can be described by an entity-cycle diagram [MATHS77, MATHS84].

Mathewson's [MATHS77, MATHS84] DRAFT system represents some of the most extensive work on program generators. DRAFT is a collection of generators which produce programs based on the entity-cycle diagram in one of five target languages. Davies' [DAVIN79] MISDESM system, based on queueing models, features different interactive questionnaires for each world view. The modeler selects the questionnaire most appropriate for his conceptual framework. Systems theory [ZEIGB84a] serves as the foundation for Subrahmanian's [SUBRE81] program generator. This generator is applicable only to simple queueing models using a next-event representation, but the work is of interest because it represents one of the first attempts to base a generator on a theory.

A program generator can be a useful tool for simulation model development because it assists the modeler in translating his conceptual model into an executable language [NANCR77]. However, its usefulness is affected by several factors. Firstly, the accuracy and completeness of the generated product often depends on a modeler's knowledge of the generator. A modeler with limited knowledge tends to utilize the generator only to produce a skeleton program to which additional code must be supplied. On the other hand, a modeler with intimate knowledge of a generator creates a program requiring few, if any, enhancements. Secondly, the target SPL influences the information extracted from the modeler during the generative inquiry. The final restriction stems from the programming perspective inherent to the entity-cycle diagram and other forms of generative discourse. A program generator deals with model development from a programming point of view.

1.3 Towards a General Theory of Simulation

Simulation programming languages and program generators focus on the simulation program **NOT** the simulation model. **To develop a general theory of simulation, attention must be concentrated on the model.** A general structure for a simulation model, consistent definitions to describe a model, and a general developmental framework need to be identified [NANCR81a, NANCR84]. Research in the area includes Lackner's **Calculus of Change**, Zeigler's **System Theoretic Approach**, and Nance's **Conical Methodology**.

Lackner [LACKM62, LACKM64] proposed the **Calculus of Change** in the early 1960's during the period that SPLs were first coming into use. The Calculus of Change advocates a general model structure when others were just beginning to discover special purpose languages for simulation. The general model structure is based on model state changes; however, no insight is provided in how to identify these state changes.

Zeigler's [ZEIGB84b] **system theoretic approach** represents some of the most extensive work towards developing a general theory of simulation. He proposes system theory as a developmental framework, and he offers some insight into the general structural properties of simulation models.

Zeigler views a model as a system which can be described statically and dynamically. A description of the static structure includes identification of subsystems, their variables, system inputs and outputs, and system states. The dynamic description consists of identifying variables, defined in the static description, which affect system state and specifying state transition functions. These descriptions are expressed in a formal language derived from set theory and logic. Essentially, a system and a model of that system can be characterized through identical formalisms.

Nance's [NANCR81a] **Conical Methodology** provides the modeler with a set of definitions, a model structure, and a developmental framework based on the object oriented paradigm (see [FAUGW80]). Lacking the rigorous but rigid representation of general systems theory, the Conical Methodology prescribes an instructive approach to model specification. A Model Development Environment based on the precepts of the methodology seeks to accomplish the following objectives: [NANCR81a, p. 22]

- Guide the modeler in structuring and organizing the model.
- Utilize an unrestrictive system to impose the axiomatic development process.
- Diagnose problems in the model early in its development.
- Produce documentation as a result of the process.
- Assist in the organization of the experimental model.

The Conical Methodology instructs the modeler to perceive a model as a collection of objects which interact through their respective attributes. A two-phase development framework of **Top-Down Definition** and **Bottom-Up Specification** is employed in the development process.

The **Top-Down Definition** phase involves the modeler's decomposing the model into its subordinate objects, then in turn decomposing these objects into their subordinates, and so on, until a point is reached where no more decomposition is warranted. As the modeler is performing this decomposition, each model object is described by a set of attributes. The Conical Methodology provides a group of definitions which enable the modeler to classify each attribute. This classification scheme gives information about the role an attribute plays in the model.

At the conclusion of the definition phase, a static model description exists and structural relationships among objects have been established. But for this description to be complete, it must include model dynamics. The dynamics are defined in the **Bottom-Up Specification** phase. The methodology suggests that the modeler select an object at the base level (a leaf of the decomposition tree), and using the object's attributes, describe **HOW** the object interacts with other model objects. This process is repeated until all model objects have been specified.

The Conical Methodology seeks to impose this model development discipline without restricting the modeler. Thus, the modeler may shift between definition and specification as the need arises. The influence of the methodology is evident in all the tools of the Model Development Environment described by Balci [BALCO86a]. A prototype of one tool, the Model Generator [HANSR84], conveys the concepts quite clearly.

In summary, the works of Nance, Zeigler, and Lackner share a common goal of separating model development from programming. To accomplish this goal, they have identified general modeling characteristics and proposed some developmental methods. Of the three approaches, only the Conical Methodology has been embedded in an environment.

1.4 Model Development Life Cycle

An essential step in the creation of a model development discipline is the identification of major developmental phases. The Model Life Cycle, as defined by Nance [NANCR81a] and Balci [BALCO86a, BALCO86b] in Figure 1, depicts these phases. An oval circle represents a phase, and a dashed line represents a process causing transition to the next phase. The diagram implies that development is a sequential procedure when in actuality it is an iterative procedure with the return to previous developmental phases expected [BALCO86b].

The Model Development Life Cycle is embodied within a more comprehensive life cycle model. The Model Life Cycle begins with **System and Objectives Definition** and ends with **Model Results.** The definition of the system and objectives represents the model requirements analogous to software requirements The Conceptual Model is the mental model the modeler has of the system. This mental model must be communicated to others in some type of representational form (see [BALCO86b, p. 8] for examples of representational forms). After the Communicative Model is verified to be an accurate reflection of the requirements, it is translated, either automatically or by a programmer, into an executable Programmed Model. Conditions for testing the model are added to the Programmed Model to form the Experimental Model, which is executed to produce Model Results.

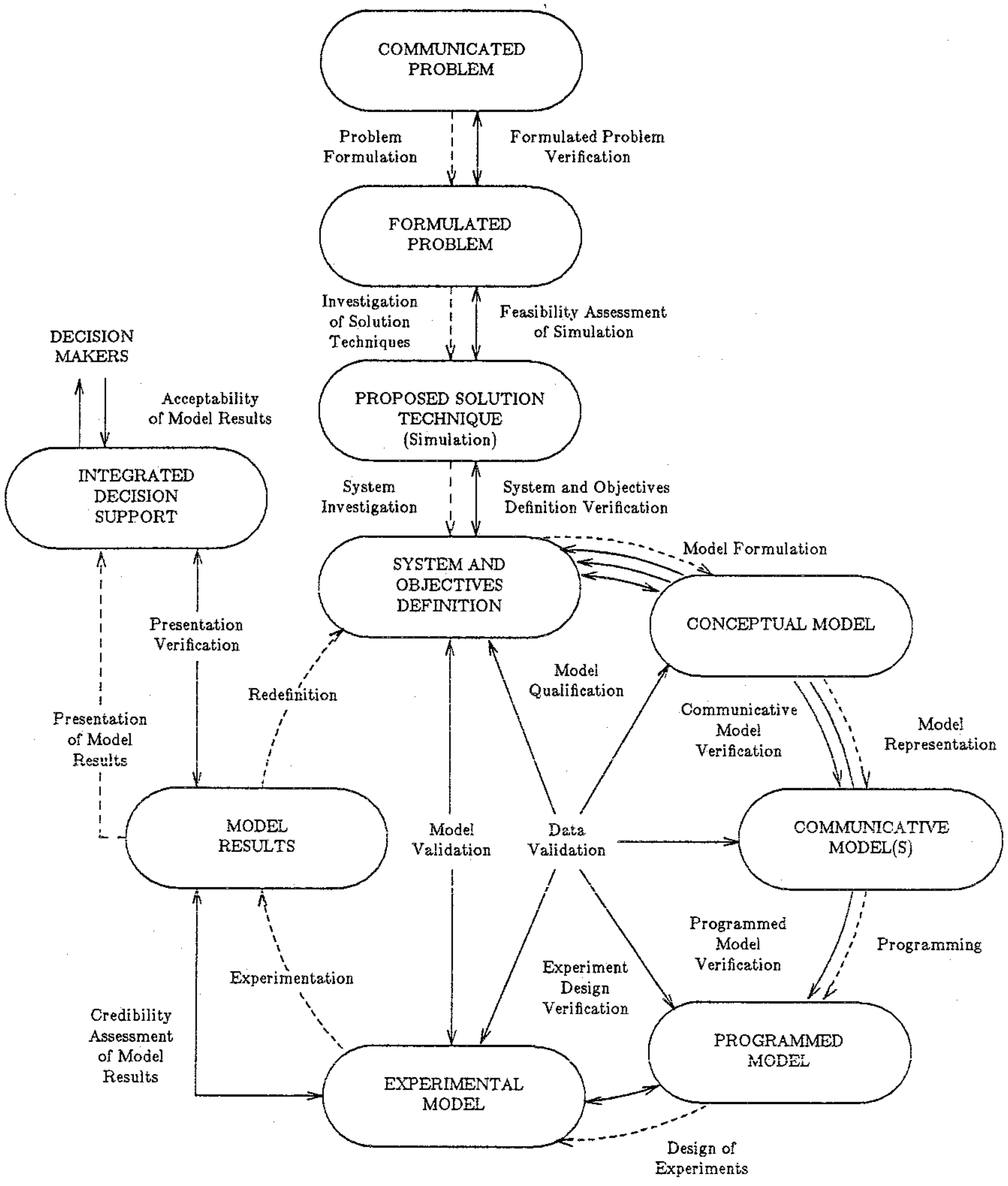
COMMUNICATED PROBLEM
Problem Formulation
Formulated Problem Verification
FORMULATED PROBLEM
Investigation of Solution Techniques
Feasibility Assessment of Simulation
PROPOSED SOLUTION TECHNIQUE (Simulation)
System Investigation
System and Objectives Definition Verification
DECISION MAKERS
Acceptability of Model Results
INTEGRATED DECISION SUPPORT
SYSTEM AND OBJECTIVES DEFINITION
Model Formulation
CONCEPTUAL MODEL
Presentation Verification
Presentation of Model Results
Redefinition
Model Qualification
Communicative Model Verification
Model Representation
MODEL RESULTS
Model Validation
Data Validation
COMMUNICATIVE MODEL(S)
Programmed Model Verification
Programming
Experimentation
Experiment Design Verification
Credibility Assessment of Model Results
PROGRAMMED MODEL
EXPERIMENTAL MODEL
Design of Experiments

Figure 1. Model Life Cycle

The Model Generator, mentioned above, assists in the transition from the Conceptual Model to the Communicative Model. The Communicative Model represents a problem-oriented model description with an emphasis on WHAT rather than HOW. Software specification languages serve a similar role in the development of software systems, and it is this similarity that motivates the examination of specification languages.

2. AN EXAMINATION OF SPECIFICATION LANGUAGES

Specification is the process of describing system behavior so as to assist the system designer in clarifying his conceptual view of the system. The purpose of specification in the development of software is to separate WHAT the system is to do from HOW it is to be done [BALZR79]. Just as with simulation models, many ways exist to describe the same software system. A specification language offers guidance in perceiving a system by characterizing and categorizing the most relevant behavior; but, more importantly, a specification language is the medium of communication for expressing this behavior.

A variety of specification languages can be found throughout the literature. In an attempt to organize this material into a coherent body, the language discussion is structured as follows:

1. Identification of specification properties,
2. Definition of software development life cycles, and
3. Classification of specification languages.

The classification scheme is based upon the software development life cycle, and the specification properties are used to determine the relative strengths and weaknesses of each language class.

2.1 Specification Properties

Table 1 summarizes the properties of a good specification and a good specification language. See [STOEJ84] for a more complete discussion of specification properties and [BALZR79] for a discussion of how the desired properties of a specification influence the design of a specification language.

2.2 Software Development Life Cycles

A specification language can be categorized according to the phase of the software development life cycle it supports. Each phase of the life cycle is responsible for collecting certain information [FREEP83] about the proposed system, and a specification language for that phase is constructed to encourage the expression of this information. Table 2 lists and defines each phase of the traditional software development life cycle [BERZV85, WASSA83].

An alternate view of the life cycle is proposed by Balzer [BALZR83] and Zave [ZAVEP84a] called the **Operational Approach.** This approach replaces functional specification, architectural design, and detailed design by an operational specification phase. An operational specification serves as an executable prototype which through a series of transformations finally becomes the implementation. The advantage of this approach is the client can experiment with the proposed system much more quickly than in the traditional life cycle [BALZR83]. The operational approach is relatively new, but a few operational specification languages have been developed such as GIST [BALZR82, GOLDN80], IORL [SIEVG85], and PAISLey [ZAVEP81, ZAVEP84b].

TABLE 1. Specification Properties

Properties of a "good" specification

- Understandable
- Appropriate for many audiences
- Presentable in varying levels of detail
- Different views of same system can be presented
- Separates implementation and description details
- Includes description of environment
- Information is localized
- Easily modifiable
- Analyzable

Properties of a "good" specification language

- Encourages modularization
- Encourages hierarchical descriptions
- Allows use of terminology of current application
- Mixture of formal and informal constructs
- Encourages use of a developmental method
- Produces documentation as a by-product
- Easy to use and learn
- Simple, precise, unambiguous syntax and semantics
- Full range of acceptable system behavior can be described
- Ability to describe variety of systems
- Provides ability to access specification completeness
- Nonprocedural

Properties of a "good" simulation specification language

- Independent of simulation programming languages
- Allows expression of static and dynamic model properties
- Facilitates model validation and verification.

REFERENCES

BALZR79, BERZV85, HANDP80, MARTJ85b, NANCR77, RIDDW79, STOEJ84

2.3 A Classification of Specification Languages

As mentioned earlier, a specification language can be classified according to the life cycle phase it best supports. The requirements definition languages, such as the diagrammatic

TABLE 2. Traditional Software Development Lifecycle

REQUIREMENTS DEFINITION • Description of problem to be solved
FUNCTIONAL SPECIFICATION • Description of behavior of a system that solves problem • Emphasis on "WHAT" system does -- system is a black box • Expression of specifier's conceptual model of system • No algorithms or data structures given • Important communication link among designers, implementers, and customers
ARCHITECTURAL DESIGN • Identify modules to perform desired system behavior • Describe module effects and interfaces — module is a black box • Definition of internal system structure • Design emphasis shifts to "HOW"
DETAILED DESIGN • Describe algorithms and data structures needed to achieve desired behavior of each module • Specific instructions on :hp1.how:ehp1. to code the system
IMPLEMENTATION • Translation of detailed design into an executable program • Testing program to see if it solves requirements
REFERENCES BERZV85, FREEP83, STOEJ84, WASSA83, YEHRT84, ZAVEP84b]

language of SADT [ROSSD77, ROSSD85], focus on expressing information needed for problem definition. These languages are of little interest for simulation model development because problem definition occurs in the model life cycle prior to the initiation of the model development phase [BALCO86a]. The detailed design languages, with their emphasis on HOW, produce a specification from which an executable program is derivable. Thus, the resulting specification may not clearly separate implementation and description details. Detailed design languages include ADA [PRIVJ83, BOOCG83], ANNA [LUCKD85], FLEX [SUTTS81], GYPSY [AMBLA77], MODEL [PRYWN79, PRYWN83, CHENT84], PDL [CAINS83], and SPECLE [BIGGT79]. Several of these languages (ADA, FLEX, and GYPSY) are also implementation languages.

Table 3 lists languages for functional specification, architectural design, and operational specification. For the evaluation of these languages, an additional classification scheme is needed. This scheme is based upon the point of view that the language gently imposes on the specifier in guiding the description of the system. The point of view is also referred to as the language's basic unit of description. In Table 3 three points of view are identified: function, object, and process. Each of these is briefly discussed.

2.3.1 Function Orientation. Specification languages that are function-oriented require that a system be defined as a series of functions which map inputs onto outputs. These languages enforce certain mathematical axioms thereby producing a *provably* correct specification [MARTJ85b]. However, the mathematical basis of these languages creates several problems:

- The resulting specification is often unreadable and difficult to understand [DAVIA82].
- The languages are hard to learn and to use for people without strong mathematical backgrounds [GEHAN82].
- The languages are applicable only for specifying small systems [WASSA83]. (AXES of Higher Order Software is an exception.)

2.3.2 Object Orientation. The object-oriented specification languages yield a model representation composed of object descriptions where each description contains the behavioral rules and characteristics for that object [FAUGW80]. Some examples of object-oriented specification languages are listed in Table 3. Each of these languages assists in the

TABLE 3. The Categorization of Specification Languages

BASIC DESCRIPTION UNIT	LANGUAGE	LIFE CYCLE PHASE	UNDERLYING MODEL	REFERENCES
FUNCTION	Algebraic Spec	F	Mathematical	[GEHAN82] [LISKB75]
	AXES	F,A,D	Mathematical	[HAMIM76] [HAMIM83] [MARTJ85a] [MARTJ85b]
	Special	F	Mathematical	[SILVB81] [STOEJ84]
OBJECT	DDN	A	Message passing	[RIDDW78a] [RIDDW78b] [RIDDW79] [RIDDW80]
	MSG.84	F,A	Message passing	[BERZV85]
	PSL/PSA	F	ERA	[STOEJ84] [TEICD77] [TEICD80] [WINTE79]
	RDL	A,D	ERA	[HEACH79] [STOEJ84]
	RML	F	ERA	[BORGA85] [OBRIP83]
	TAXIS	A,D	ERA	[BORGA85] [OBRIP83]
PROCESS	GIST	O	Stimulus response ERA	[BALZR80] [BALZR82] [FEATM83] [GOLDN80] [LONDP82]
	PAISLey	O	Stimulus response	[YEHRT84] [ZAVEP79] [ZAVEP82] [ZAVEP84a] [ZAVEP84b]
	RSL	F,O	Stimulus response	[ALFOM77] [ALFOM85] [BELLT77] [DAVIC77] [SCHEP85] [STOEJ84]

production of object descriptions, but the exact form of the object description varies with the underlying model of the language. Two models commonly employed are the Entity-Relation-Attribute (ERA) model and the message-passing model.

The ERA model [STOEJ84] views a system as being composed of entities (objects), entity attributes, and relationships among entities. ERA-based languages are widely used for applications where static descriptions are necessary (such as databases), but they have limited applicability in cases where dynamic descriptions are required. They emphasize the flow of data through system objects rather than the interactions among these objects.

Languages using a message-passing model [ROBSD81a, ROBSD81b, MCARD84], unlike the ERA-based languages, are able to express system dynamics through a series of message communications. In this model, each object is defined by its attributes and the operations it can perform. Thus, for object A to interact with object B, object A SENDS object B a message requesting that object B perform an operation. Object B RECEIVES the message and performs the requested operation, which may include sending a message to another object or back to object A. This SEND and RECEIVE format of object interaction has potential for allowing description of some of the concepts needed in simulation models such as timing constraints, concurrency, synchronization, and environmental interaction.

The choice between an ERA-based or a message-based language is very dependent on the type of system to be specified. Both models encapsulate system behavior according to objects thereby easily localizing specification information. But the major advantage of using the object-oriented approach is that the resulting specification corresponds directly and naturally to the real system [BORGA85, p. 85].

2.3.3 Process Orientation. In a process-oriented specification language, a system is decomposed into its processes where a process may represent an object or an activity. Both static and dynamic descriptions of the system are possible. The static properties of each process are defined by enumerating its possible states, inputs, and outputs [YEHRT84, ALFOM85]. The dynamics of each process are described as a series of state transitions with each state transition producing certain responses and yielding a new process state. The resulting specification shows both data flow and control flow [ZAVEP82] as well as the relationship of each process to its environment. Each process is assumed to operate in parallel with and asynchronous to the other processes [ZAVEP84b].

The origins of many process-oriented specification languages can be traced back to the need to specify operating systems. Thus, these languages are well suited for applications involving complex controls and embedded systems [ZAVEP82]. Also the underlying stimulus-response model is actually a modified finite state machine thereby making it possible to use results from finite automata theory in analyzing the specification [ALFOM85].

The process-oriented languages listed in Table 3 have several weaknesses:

- They employ excessive formal notation that is difficult to understand and learn.
- They may be difficult to apply to systems other than the types mentioned in the previous paragraph.
- The resulting specification may be neither naturally hierarchical nor very modular [ALFOM85, ZAVEP82].
- The languages, with the exception of RSL, are still in the development stages.

In conclusion, a review of the literature on software specification languages reveals a variety of approaches to specification and numerous formalisms. Several desirable properties of a specification and a specification language have been identified as well as the relationship of particular languages to phases of the software development lifecycle. Most of the languages reviewed are developed for use in certain application domains; thus, they prove easier to use and produce a better specification when applied to problems in these domains [ZAVEP84b]. This fact suggests that a more narrow examination of simulation model specification languages is necessary.

3. SIMULATION MODEL SPECIFICATION AND DOCUMENTATION LANGUAGES

As the focus in simulation model development has shifted to the model and away from the program [NANCR84, ZEIGB84c], perception of the need for simulation model specification and documentation languages (SMSDLs) has emerged. Nance [NANCR77] gives the following reasons why a SMSDL is needed:

- Model documentation must be an inseparable part of model development, and during the early stages of model development, model specification and model documentation are identical. A SMSDL can encourage model documentation.
- A SMSDL can help bridge the communication gap between the model developer and the customer.
- A SMSDL can provide a set of terms which leads to a precise model description, while enabling sufficient generality, and producing a hierarchical specification which is independent of current SPLs.

Even though these SMSDLs are domain dependent, the properties listed in Table 1 are still very applicable to them. The biggest challenge in the design of any SMSDL is HOW to express model dynamics [NANCR77] because it is the dynamic dependencies in a simulation model which make it inherently complex [NANCR84]. Some examples of SMSDLs are DELTA, GEST, ROSS, a SIMULA derived SMSDL proposed by Frankowski and Franta, and the Condition Specifications. Each of these is briefly discussed.

3.1 DELTA

The DELTA language [HOLBE77, HANDP80] represents one of the most extensive SMSDL efforts, and it can be characterized as an object-oriented specification language for describing complex systems. The language uses a mixture of formal and informal constructs to describe a system as a series of objects, their attributes, their states, and the actions each object performs. The language encourages the production of a specification which is appropriate for communicating details of system behavior to an audience with diverse backgrounds. Its SIMULA-like constructs enable the expression of simulation model dynamics. Specifically, one can describe time, events, time-consuming actions, instantaneous actions, parallel actions, state changes, and activity interruptions. However, the literature suggests that DELTA has not actually been applied to simulation development [NANCR81a] nor is it part of an automated system. In fact, the literature provides little information about DELTA, and its current state is unknown.

Initially, the developers of DELTA had planned to design three languages:

- a general systems description language, DELTA
- a high level programming language, BETA
- and a systems programming language, GAMMA.

As of 1980, only DELTA exists, and no mention is made of GAMMA. The developers plan to use DELTA as the overall system development language and to translate the DELTA system specification into the executable programming language BETA.

3.2 GEST

GEST [ORENT79, ORENT84], based on the principles of system theory, provides a conceptual framework for specifying discrete, continuous, and memoryless models. The static model structure consists of the definition of input, output, and state variables; the dynamic model structure is described via a series of state transition functions. A GEST specification also incorporates Zeigler's [ZEIGB84c] experimental frame concept. An experimental frame is the specification of the data a simulation model is expected to produce in order to answer the questions posed in the study objectives [ZEIGB84c, p. 22]. Information specified in a GEST experimental frame includes such items as the basic unit of time, simulation termination conditions, state variable initializations, and data collection details [ORENT84, p. 316]. Future plans are for GEST to be integrated into a computer-assisted modeling system [ORENT84].

3.3 ROSS

ROSS [KLAHP80, FAUGW80, MCARD81, MCARD84], developed by the RAND Corporation, represents one of the first attempts to combine artificial intelligence and simulation. It is designed for developing interactive knowledge-based event-driven simulators for particular applications. Currently, two combat-oriented simulators (SWIRL [KLAHP82] and TWIRL [KLAHP84]) have been developed. ROSS is not a specification language, but rather it is a LISP-based system which supports the development of the above mentioned simulators. It is of interest because it is a good example of of the use of the object-oriented message-passing paradigm, demonstrating how concepts from artificial intelligence and simulation can be combined.

3.4 SMSDL — Frankowski and Franta

Frankowski and Franta [FRANE80] propose a process-oriented simulation model specification and documentation language for specifying discrete-event simulations. In this SMSDL, the model element (or object) is the primary unit of specification. Each element is described by attributes, axioms, and a scenario. Of these three, the scenario is of most interest because it contains the details of an element's behavior throughout the life of the model.

The scenario describes the changes of state in the model which trigger each element's behavior. These changes of state may be invoked by the element itself or other model elements. The SMSDL does not differentiate between whether a behavior is dependent on a change in state or a change in time. Time is simply viewed as an attribute which may produce a state change. The behaviors that are specified in a scenario are referred to as actions. Constructs are available in the SMSDL for expression of interruptible and noninterruptible actions, actions that have duration, concurrent actions, and actions that occur in a predefined order.

These constructs, as well as the constructs for expressing conditions, reflect the influence of the language SIMULA on the design. This SIMULA influence is a disadvantage for the SMSDL by its advocacy of a particular world view. In fact, Frankowski and Franta suggest that the transition between specification and program can be eased if the program is coded in a language akin to SIMULA [FRANE80, p. 725].

Although the SMSDL suffers from a world view influence, it represents an attempt to produce a readable, English-like specification which is suitable for communicating with a diverse audience. But more importantly, the SMSDL addresses some of the difficulties in expressing model dynamics and offers some possible solutions.

3.5 Condition Specifications

Another SMSDL has been suggested by Overstreet [OVERM82, OVERM85] called the

Condition Specifications (CS). The CS overcomes a major shortcoming of the previously mentioned SMSDL which is the inability to produce a world view independent specification. In addition to the property of world view independence, a CS has the following properties: [OVERM82, p. 5]

- Independence from implementation languages
- Creates an analyzable specification
- Assists in detecting errors in the specification
- Permits translation to a specification in any of the three traditional world views
- Enables any discrete-event simulation model to be completely described
- Encourages successive refinement and elaboration of the specification
- Produces a specification in which the model elements have direct correspondence to the elements of the modeled system.

A Condition Specification consists of three components [OVERM85, p. 196]. The *interface specification* identifies the input and output attributes of the model, and it is through these attributes that the model communicates with its environment. The *report specification* describes the data that is to be produced as a result of an execution of the simulation. The *specification of model dynamics* is the most important CS component. This component includes a set of object specifications for describing the static model structure and a set of transition specifications for describing the dynamic model structure. An object specification exists for each object in the model. This specification names the object and defines all the attributes associated with that object. The transition specifications describe the changes that occur in the model as a series of Condition Action Pairs (CAPs). A CAP is simply a rule composed of a boolean condition and an action to be performed whenever the condition is true. A boolean condition may be based upon time, state, or both. An action represents an object's response to the boolean condition and may include such responses as changing the value of an object attribute, scheduling another action to occur in the future, or terminating the simulation. The specific notation for expressing a CAP is described in [OVERM85].

The Condition Specifications provide guidance on how to build a model and how to describe the dynamic relationships in a model [OVERM85] by using an object-oriented rule-based approach. They represent a precise statement of the time and state relationships [NANCR81b]. However, the CS formalism is very similar to a programming language [HANSR84]; thus, it may be difficult for a modeler with a limited computer science background to use. For this reason, a modeler is not intended to directly specify a model in the CS notation. Rather, the modeler is to interact with a tool, a model generator, which will automatically produce a model specification in the CS formalism [OVERM84].

4. SUMMARY AND CONCLUSIONS

This comprehensive review of the literature presents the breadth and diversity of views of *specification* held by researchers in simulation modeling and the software engineering community. While the indication that the former group is aware of the activities and contributions of the latter, the reverse is *not* apparent. Consequently, both simulation modeling and the domain of embedded, real-time software development continues to struggle with only partial understanding of the difficulties in specifying dynamic behavior and providing precise capabilities for specifying object interactions.

Based on the review presented above, several conclusions with respect to the role of system specification techniques are warranted:

- The function of a methodology in the specification process is highly dependent on the phase of the software life cycle being addressed.
- No clear perception of the role of specification appears in either the software or simulation communities, but the object oriented paradigm clearly dominates the latter.

- With a few exceptions, development environments and the tools that comprise them are rarely tied to an undergirding methodology; rather they appear to be limited to application domains and evince a degree of ad hoc design.

REFERENCES

ALFOM77 Alford, M. W. A Requirement Engineering Methodology for Real-time Processing Requirements, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 60-69.

ALFOM85 Alford, M. W. SREM at the Age of Eight: The Distributed Computing Design System, *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 36-46.

AMBLA77 Ambler, A., et al. GYPSY - A Language for Specification and Implementation of Verifiable Programs, *Proceedings of the Conference on Language Design for Reliable Software, ACM SIGPLAN Notices*, Vol. 12, No. 3, March 1977, pp. 1-10.

BALCO86a Balci, Osman. Requirements for Model Development Environments, *Computers and Operations Research*, Vol. 13, No. 1, January - February 1986, pp. 53-67.

BALCO86b Balci, Osman. Guidelines for Successful Simulation Studies, Technical Report TR-85-2, Department of Computer Science, Virginia Tech, Blacksburg, May 1986.

BALZR79 Balzer, R. and N. Goldman. Principles of Good Software Specification and Their Implications for Specification Languages, *Proceedings IEEE Conference on Specification for Reliable Software*, April 1979, pp. 58-67.

BALZR80 Balzer, Robert. An Implementation Methodology for Semantic Database Models, in *Entity-Relationship Approach to Systems Analysis and Design*, P.P. Chen, ed., North-Holland Publishing Company, Amsterdam, 1980, pp. 433-444.

BALZR82 Balzer, Robert M., Neil M. Goldman and David S. Wile. Operational Specification as the Basis for Rapid Prototyping, *ACM SIGSOFT Software Engineering Notes*, Vol. 7, No. 5, December 1982, pp. 3-16.

BALZR83 Balzer, Robert, Thomas E. Cheatham, Jr., and Cordell Green. Software Technology in the 1990's: Using a New Paradigm, *IEEE Computer*, Vol. 16, No. 11, November 1983, pp. 39-45.

BELLT77 Bell, T.E., D.C. Bixler and M.E. Dyer. An Extendable Approach to Computer-aided Software Requirements Engineering, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 49-60.

BERZV85 Berzins, Valdis. Analysis and Design in MSG.84: Formalizing Functional Specifications, *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 8, August 1985, pp. 657-670.

BIGGT79 Biggerstaff, T.J. The Unified Design Specification System (UDS), *Proceedings IEEE Conference on Specification for Reliable Software*, April 1979, pp. 104-118.

BOOCG83 Booch, Grady. Object-Oriented Design, in *Tutorial on Software Design Techniques*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, pp. 420-436.

BORGA85 Borgida, Alexander, Sol Greenspan and John Mylopoulos. Knowledge Representation as the Basis for Requirements Specifications, *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 82-91.

CAINS83 Caine, Stephen H. and E. Kent Gordon. PDL - A Tool for Software Design, in *Tutorial on Software Design Techniques*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 485-490.

CHENT84 Cheng, Thomas T., Evan D. Lock, and Noah S. Prywes. Use of Very High Level Languages and Program Generation by Management Professionals, *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September 1984, pp. 552-563.

DAVIN79 Davies, N.R. Interactive Simulation Program Generation, in *Methodology in Systems Modelling and Simulation*, B.P. Zeigler, M.S. Elzas, G.J. Klir and T.I. Oren, eds., North-Holland Publishing Company, Amsterdam, 1979, pp. 179-200.

DAVIA82 Davis, A.M. The Design of a Family of Application-Oriented Requirements Languages, *IEEE Computer*, Vol. 15, No. 5, May 1982, pp. 21-28.

DAVIC77 Davis, C.G. and C.R. Vick. The Software Develoment System, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 69-84.

FAUGW80 Faught, W.S., P. Klahr and G.R. Martins. An Artificial Intelligence Approach to Large-Scale Simulation, *Summer Simulation Conference*, Seattle, Washington, August 25-27, pp. 231-235.

FEATM83 Feather, Martin S. Reuse in the Context of a Transformation Based Methodology, *Proceedings of the Workshop on Reusability in Programming*, Newport, Rhode Island, September 7-9, 1983, pp. 50-58.

FRANE80 Frankowski, E.N. and W.R. Franta. A Process Oriented Simulation Model Specification and Documentation Language, *Software -- Practice and Experiences*, Vol. 10, No. 9, September 1980, pp. 721-742.

FREEP83 Freeman, Peter. Fundamentals of Design, in *Tutorial on Software Design*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 2-22.

GEHAN82 Gehani, Narain. Specifications: Formal and Informal -- A Case Study, *Software - Practice and Experience*, Vol. 12, 1982, pp. 433-444.

GOLDN80 Goldman, N. and D. Wile. A Relational Database Foundation for Process Specification, in *Entity-Relationship Approach to Systems Analysis and Design*, P.P. Chen, ed., North-Holland Publishing Company, Amsterdam, 1980, pp. 413-432.

HAMIM76 Hamilton, Margaret and Saydean Zeldin. Higher Order Software - A Methodology for Defining Software, *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, January 1976, pp. 9-32.

HAMIM83 Hamilton, M. and S. Zeldin. The Relationship Between Design and Verification, in *Tutorial on Software Design Techniques*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 641-668.

HANDP80 Handlykken, Peter and Kristen Nygaard. The DELTA System Description Language Motivation, Main Concepts, and Experience from Use, in *Software Engineering Environments*, Horst HUNKE, ed., North-Holland Publishing Company, Amsterdam, 1980, pp. 173-190.

HANSR84 Hansen, Robert Hans. The Model Generator: A Crucial Element of the Model Development Environment, Technical Report CS84008-R, Department of Computer Science, Virginia Tech, Blacksburg, August 1984.

HEACH79 Heacox, H.C. RDL - A Language for Software Development, *ACM SIGPLAN Notices*, Vol. 14, December 1979, pp. 71-79.

HENRJ83 Henriksen, James O. Event List Management - A Tutorial, *Proceedings of the 1983 Winter Simulation Conference*, Arlington, Virginia, December 12-14, 1983, pp. 542-551.

HOLBE77 Holbaek-Hanssen, E., P. Handlykken and K. Nygaard. *System Description and the Delta Language*, Report No. 4, Norwegian Computing Center, Oslo, 1977.

KIVIP69 Kiviat, P.J. Digital Computer Simulation: Computer Programming Languages, *RAND Corp. Memorandum* RM-5883-PR, January 1969.

KLAHP80 Klahr, P., and W.S. Faught. Knowledge-Based Simulation, *Proceedings of the First Annual Conference of the American Association for Artificial Intelligence*, Stanford, California, 1980, pp.181-183.

KLAHP82 Klahr, Philip, David McArthur, Sanjai Narain and Eric Best. *SWIRL: Simulating Warfare in the ROSS Language*, Rand Report N-1885-AF, The Rand Corporation, Santa Monica, California, September 1982.

KLAHP84 Klahr, Philip, et al. *TWIRL: Tactical Warfare in the ROSS Language* Rand Report R-3158-AF, The Rand Corporation, Santa Monica, California, October 1984.

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

LACKM64 Lackner, Michael R. Digital Simulation and System Theory, Systems Development Corporation SP-1612, Santa Monica, California, April 6, 1964.

LISKB75 Liskov, Barbara H. and Stephen N. Zilles. Specification Techniques for Data Abstraction, *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 1, March 1975, pp. 7-19.

LONDP82 London, P.E. and M.S. Feather. Implementing Specification Freedoms, in *Science of Computer Programming*, Vol. 2, North-Holland Publishing Company, Amsterdam, 1982, pp. 91-131.

LUCKD85 Luckham, David C. and Friedrich W. von Henke. An Overview of Anna, a Specification Language for ADA, *IEEE Software*, Vol. 2, No. 2, March 1985, pp. 9-22.

MARTJ85a Martin, James and Carma McClure. *Diagramming Techniques for Analysts and Programmers*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

MARTJ85b Martin, James. *System Design From Provably Correct Constructs*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

MATHS77 Mathewson, S.C. and J.H. Allen. Draft/GASP -- A Program Generator for GASP, *Proceedings Tenth Annual Simulation Symposium*, Tampa, Florida, 1977, pp. 211-225.

MATHS84 Mathewson, S.C. The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling, *IIE Transactions*, Vol. 16, No. 1, March 1984, pp. 3-18.

MCARD81 McArthur, D. and H. Sowizral. An Object-Oriented Language for Constructing Simulations, *Proceedings of the International Joint Conference on Artificial Intelligence*, Vancouver, Canada, 1981, pp. 809-814.

MCARD84 McArthur, David, Philip Klahr and Sanjai Narain. *ROSS: An Object-Oriented Language for Constructing Simulations*, Rand Report R-3160-AF, The Rand Corportion, Santa Monica, California, December 1984.

NANCR77 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, Blacksburg, June 1977.

NANCR81a Nance, Richard E. Model Representation in Discrete Event Simulation: The Conical Methodology, Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, March 15, 1981.

NANCR81b Nance, Richard E. The Time and State Relationships in Simulation Modeling, *Communications of the ACM*, Vol. 24, No. 4, April 1981, pp. 173-179.

NANCR84 Nance, Richard E. Model Development Revisited, *Proceedings of the 1984 Winter Simulation Conference*, Dallas, Texas, November 28-30, 1984, pp. 75-80.

OBRIP83 O'Brien, Patrick D. An Integrated Interactive Design Environment for TAXIS, *SOFTFAIR*, Arlington, Virginia, July 25-28, 1983, pp. 298-306.

ORENT79 Oren, Tuncer I. and Bernard P. Zeigler. Concepts for Advanced Simulation Methodologies, *Simulation*, Vol. 32, No. 3, March 1979, pp. 69-82.

ORENT84 Oren, Tuncer I. GEST - A Modelling and Simulation Language Based on System Theoretic Concepts, in *Simulation and Model-Based Methodologies: An Integrative View*, T.I. Oren, B.P. Zeigler, M.S. Elzas, eds., Springer-Verlag, New York, 1984, pp. 281-335.

OVERM82 Overstreet, C. Michael. *Model Specification and Analysis for Discrete Event Simulation*, PhD Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, December 1982.

OVERM84 Overstreet, C. Michael and Richard E. Nance. Graph-Based Diagnosis of Discrete Event Model Specifications, Technical Report CS83028-R, Department of Computer Science, Virginia Tech, Blacksburg, June 1984.

OVERM85 Overstreet, C.M. and R. Nance. A Specification Language to Assist in Analysis of Discrete Event Simulation Models, *Communications of the ACM*, Vol. 28, No. 2, February 1985, pp. 190-201.

OVERM86 Overstreet, C.M. and R.E. Nance. World View Based Discrete Event Model Simplication, in *Modeling and Simulation Methodology in the Artificial Intelligence Era*, M.S. Elzas, T.I. Oren, and B.P. Zeigler, eds., North-Holland Publishing Company, to appear.

PRIVJ83 Privitera, Dr. J.P. ADA Design Language for the Structured Design Methodology, in *Tutorial on Software Design Techniques*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 491-505.

PRYWN79 Prywes, N.S., A. Pnueli and S. Shastry. Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development, *ACM Transactions on Programming Languages and Systems*, Vol. 1, No. 2, October 1979, pp. 196-217.

PRYWN83 Prywes, Noah S. and Amir Pnueli. Compilation of Nonprocedural Specifications into Computer Programs, *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 3, May 1983, pp. 267-279.

RIDDW78a Riddle, W.E., et al. A Descriptive Scheme to Aid the Design of Collections of Concurrent Processes, *Proceeding AFIPS National Computer Conference*, Anaheim, California, June, 1978, pp. 549-554.

RIDDW78b Riddle, W.E., et al. Behavior Modeling During Software Design, *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 4, July 1978, pp. 283-292.

RIDDW79 Riddle, William E. An Event-Based Design Methodology Supported by DREAM, in *Tutorial on Software Design Techniques*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 378-392.

RIDDW80 Riddle, William E. An Assessment of DREAM, in *Software Engineering Environments*, Horst HUNKE, ed., North-Holland Publishing Company, Amsterdam, 1980, pp. 191-221.

ROBSD81a Robson, David. Object-Oriented Software Systems, *BYTE*, Vol. 6, No. 8, August 1981, pp. 74-86.

ROBSD81b Robson, David and Adele Goldberg. The Smalltalk-80 System, *BYTE*, Vol. 6, No. 8, August 1981, pp. 36-48.

ROSSD77 Ross, Douglas T. and Kenneth E. Schoman, Jr. Structured Analysis for Requirements Definition, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 6-15.

ROSSD85 Ross, Douglas T. Applications and Extensions of SADT, *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 25-34.

SCHEP85 Scheffer, Paul A., Albert H. Stone III and William E. Rzepka. A Case Study of SREM, *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 47-54.

SIEVG85 Sievert, Gene E. and Terrence A. Mizell. Specification-Based Software Engineering with TAGS, *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 56-65.

SILVB81 Silverberg, Brad A. An Overview of the SRI Hierarchical Development Methodology, *Software Engineering Environments*, Horst HUNKE, ed., North-Holland Publishing Company, Amsterdam, 1981, pp. 235-252.

STOEJ84 Stoegerer, J.K. A Comprehensive Approach to Specification Languages, *The Australian Computer Journal*, Vol. 16, No. 1, February 1984, pp. 1-13.

SUBRE81 Subrahmanian, E. and R.L. Cannon. A Generator Program for Models of Discrete-Event Systems, *Simulation*, Vol. 36, No. 3, March 1981, pp. 93-101.

SUTTS81 Sutton, S.A. and V.R. Basili. The FLEX Software Design System: Designers Need Languages, Too, *IEEE Computer*, Vol. 14, No. 11, pp. 95-102.

TEICD77 Techroew, D. and E.A. Hershey III. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems, *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January 1977, pp. 41-48.

TEICD80 Teichrow, D., et al. Application of the Entity-Relationship Approach to Information Processing Systems Modeling, in *Entity-Relationship Approach to Systems Analysis and Design*, P.P. Chen, ed., North-Holland Publishing Company, Amsterdam, 1980, pp. 15-38.

WASSA83 Wassermann, Anthony I. Information System Design Methodology, in *Tutorial on Software Design*, Peter Freeman and Anthony Wassermann, eds., IEEE Computer Society Press, 1983, pp. 43-62.

WINTE79 Winters, E.W. An Analysis of the Capabilities of PSL: A Language for System Requirements and Specifications, *IEEE COMPSAC*, Chicago, Illinois, November, 1979, pp. 283-288.

YEHRT84 Yeh, Raymond T., Pamela Zave, Alex Paul Conn and George E. Cole, Jr. Software Requirements: New Directions and Perspectives, in *Handbook of Software Engineering*, C.R. Vick and C.V. Ramamoorthy, eds., Van Norstrand Reinhold Company, New York, 1984, pp. 519-543.

ZAVEP79 Zave, Pamela. A Comprehensive Approach to Requirements Problems, *IEEE COMPSAC*, Chicago, Illinois, November 1979, pp. 117-122.

ZAVEP81 Zave, Pamela and Raymond T. Yeh. Executable Requirements for Embedded Systems, *Proceedings of the Fifth International Conference on Software Engineering, IEEE*, San Diego, California, March 9-12, 1981, pp. 295-304.

ZAVEP82 Zave, Pamela. An Operational Approach to Requirements Specification for Embedded Systems, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 3, May 1982, pp. 250-269.

ZAVEP84a Zave, Pamela. The Operational Versus the Conventional Approach to Software Development, *Communications of the ACM*, Vol. 27, No. 2, February 1984, pp. 104-118.

ZAVEP84b Zave, Pamela. An Overview of the PAISLey Project - 1984, *ACM SIGSOFT Software Engineering Notes*, Vol. 9, No. 4, July 1984, pp. 12-19.

ZEIGB84a Zeigler, Bernard P. Multifaceted Modeling Methodology: Grappling with the Irreducible Complexity of Systems, *Behavioral Science*, Vol. 29, No. 3, 1984, pp. 169-178.

ZEIGB84b Zeigler, Bernard P. System-Theoretic Representation of Simulation Models, *IIE Transactions*, Vol. 16, No. 1, March 1984, pp. 19-34.

ZEIGB84c Zeigler, Bernard P. Theory and Application of Modeling and Simulation: A Software Engineering Perspective, in *Handbook of Software Engineering*, C.R. Vick and C.V. Ramamoorthy, eds., Van Nostrand Reinhold Company, New York, 1984, pp. 1-25.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER SRC-86-005 / TR-86-19 Dept. CS	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Simulation Model Development: System Specification Techniques		5. TYPE OF REPORT & PERIOD COVERED Final Report: June 1985 - August 1986
		6. PERFORMING ORG. REPORT NUMBER SRC-86-005 / CS TR-86-19
7. AUTHOR(s) Lynne F. Barger and Richard E. Nance		8. CONTRACT OR GRANT NUMBER(s) N60921-83-G-A165 B001-Z
9. PERFORMING ORGANIZATION NAME AND ADDRESS Systems Research Center and Department of Computer Science Virginia Tech, Blacksburg, VA 24061		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Sea Systems Command SEA61E Washington, D.C. 20362		12. REPORT DATE 15 August 1986
		13. NUMBER OF PAGES 31
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* Naval Surface Weapons Center Dahlgren, VA 22448		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This report is distributed to scientists and engineers at the Naval Surface Weapons Center and is made available on request to other Navy scientists. A limited number of copies is distributed for peer review.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

To Navy research and development centers and university-based laboratories.

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Programming Languages, formal definitions and theory; Simulation and Modeling, simulation languages. Simulation model specification, model documentation, language categories, model development environments.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A review of software specification techniques and specification languages is described. Special emphasis is given to simulation model specification and the degree to which general software techniques are applicable in the modeling domain. The role of specifaction is examined in terms of existing techniques as well as abstraction permitting the identification of desirable properties unrelated to existing tools. A categorization of specification languages assists in understanding the similarities and differences among current approaches. Important conclusions are

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)* that: (1) specification methodologies are highly dependent on the phase of the software life cycle, (2) both the general software and simulation communities lack a clear perception of the role of specification, and (3) tools and environments tend to lack an underlying methodological foundation.