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

A New Formalism for Discrete Event Simulation

Ashvin Radiya
Robert G. Sargent
Simulation Research group
441 Link Hall
Syracuse University
Syracuse, New York 13244

Abstract

A new formalism for discrete event simulation is proposed based on the theoretical frameworks used for the formal analysis and specification of the different aspects of computer languages. The major elements of such a formalism are identified, defined, and discussed. This formalism includes the specification and validation of the structural and behavioral aspects of models specified in it. In this formalism, the behavioral specifications can be used to synchronize different components of a model. This enhances the modeling methodology.

1. Introduction

Roughly, a formalism formalizes "how and what" can be expressed about a problem entity. The *problem entity* is a system (real or proposed), idea, situation, policy, or a phenomena to be modeled. In this paper, we are concerned with a new formalism for discrete event simulation (DES). This formalism includes the following six elements: a model specification language; its semantics; model - its structure and behavior; specification of validity (correctness) of a model; proof system for model correctness; and the underlying modeling methodology. The elements of this formalism and their relationship to each other are shown in Figure 1. As shown in Figure 1, a model is specified using the primitives and constructs of the model specification language of the formalism. A model has structure and its behavior is generated by executing it. Model validity is concerned with how well a model represents a problem entity for its intended purpose. A suitable proof system must be provided that has the capability of proving the validity of a specified model behavior. The proof system must be sound with respect to the semantics of the model specification language. The modeling methodology underlying such a formalism depends on the language, usage and specification of model behavior, and the proof system. Obviously, the ambitious task of specifying a useful and complete formalism is enormous.

In this paper, the general definitions of the elements essential for specifying a formalism are considered first. Next, a particular formalism is proposed for modeling based on the techniques of DES. Only some of the elements like a model specification language, a framework of semantic concepts, and model behavior are considered. The informal definitions of the elements of the proposed formalism are considered. The precise mathematical definitions are left for the future work.

2. Elements of a Formalism

The six elements necessary for defining a formalism for discrete event simulation are defined and discussed next.

2.1 A Model Specification Language

In computer based simulation, a modeler specifies a model of a real system in a model specification language (in this paper it is referred to simply as a 'language') to achieve the desired goal. A language can be specified by giving its syntax using BNF [Backus 1959] or syntax diagrams. Different, but less expressive, languages can be obtained by suitably restricting the syntactic constructs of a given language. In this sense, the syntax of a given language contains the syntax of several less expressive languages. A language is characterized by a set of atomic actions (primitives) and constructs. These primitives and constructs are used by a modeler to specify a model. The kind of knowledge that is conveniently expressible and its form depend on the language that is used to specify a model. The syntax of a given language determines the structure or form of a model specified in that language. Different constructs have different meaning and usage for the purpose of modeling. A language (specified using syntax only) may contain inconsistencies, ommissions, and ambiguities. This deficiency in a language definition can be removed by specifying the precise mathematical meaning (semantics) of the primitives and constructs of a language.

2.2 Semantics

The semantics of a language is specified by defining mappings, in an appropriate mathematical model, from the syntactic constructs of the object language into the framework of semantic concepts. Despite the complexity and variety exhibited by programming languages, it has been shown that a framework of a remarkably small number of fundamental semantic concepts provide an adequate conceptual basis for defining concise formal models of their meaning [Tennent 1976]. The semantics of a language provides a precise and abstract meaning of a language. Some of the computational or mathematical properties of models can be expressed using the semantic concepts of the language.

A framework of semantic concepts can be specified by prescribing a collection of appropriate sets and functions that satisfy suitable properties (constraints). For example, the DEVS-formalism [Zeigler 1976] based on systems and set-theoretic concepts can be used to give semantics of DES programming languages. A formal model of a meaning of a language can be useful to a modeler as it provides a small number of concepts necessary to define the semantics. The concepts of semantics are related to the concepts of a modeling methodology which is used to construct models in the language under consideration. Direct studies of these language-independent formalisms may allow them to be appropriately extended and lead to the development of new methodologies for constructing models.

2.3 Model - its Structure and Behavior

A *model* is a modeler's representation of a problem entity expressed in a suitable model specification language to achieve a desired goal. Two important aspects of a model are its structure and its behavior. The *model structure* refers to the particular form taken by a model specification. The *model behavior* is generated when a model is executed. The model behavior must be distinguished from the model structure which refers to the particular form that a model takes. Essentially, a model is characterized by its structure or form and model behavior is generated when the model is executed.

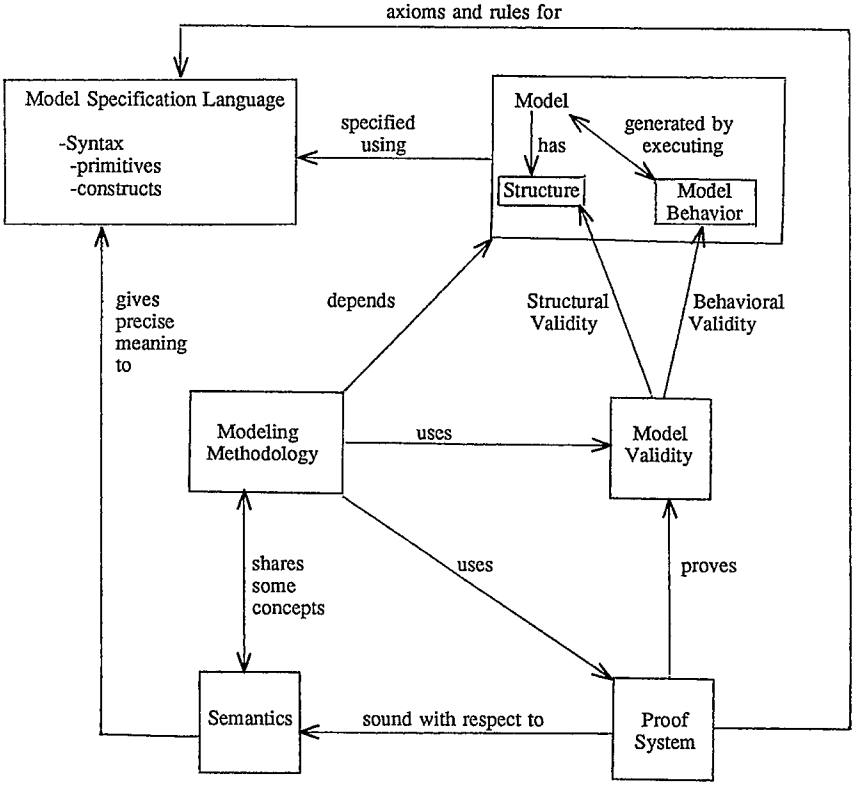

Figure 1: The Elements of the Formalism and Their Relationships

The resemblance of a model structure to the organization and form of modeler's knowledge of a problem entity increases the expressibility and readability of a model. A model should be constructed such that the model's behavior agrees with the behavior of the problem entity. Traditionally, model's behavior is specified by the trajectories of some of the variables of a model (see Zeigler (1976) for an example). This notion of model behavior is based on the assumption that a model can be specified using variables denoting values like numbers. However, interesting model behaviors can be specified differently. For example, a modeler may have made the observation: 'if events A and B occur together with events C or D then event G occurs'. A useful behavior of a model need not be restricted to variables.

A general definition of model behavior would be useful. Such a definition must include the traditional behavioral specification in terms of the properties of certain variable trajectories as well as other useful behaviors specified in terms the properties of sequences of events, activities, etc. may satisfy. In section 3.3, an attempt is made to give a definition for a general model behavior. This definition of model behavior can be used in model specification and model validation. Behavioral expressions can be defined to achieve model synchronization and specify validity conditions.

2.4 Specification of Model Validity

Model validity is concerned with how well a model represents a problem entity for its intended purpose. Two aspects of model give two major types of validity: structural and behavioral validity. *Structural validity* is concerned with the extent of agreement between the model's structure and the structure of a problem entity. *Behavioral validity* is concerned with the extent to which the model's behavior corresponds to problem entity's behavior.

As mentioned in the previous subsection, the modeling methodology is enhanced when expressions denoting behaviors can be used in model specification. Similarly, the validation techniques can also be enhanced by considering complex behaviors in addition to validating the properties of certain variable trajectories during a model execution. For example, *event validity* compares the events of occurrences of the simulation model with those of the real system. An example illustrating more complex behavioral pattern is, 'a model is valid only if activity A (specified by an initial and final event) never overlaps activity B'. To develop a complete theory of model's validity, a specification language needs to be developed to specify properties of model behavior. For example, the language of assertions based on first order predicate calculus is developed to specify the correctness of sequential programs [Hoare 1973].

2.5 Proof System

A proof system is a programming logic for reasoning about some of the properties of a model. It is an extension of predicate logic, and thus contains all the formulas, axioms, and inference rules of predicate logic. In addition to the axioms and inference rules of predicate logic, the programming logic has one axiom or inference rule for each type of statement, as well as some statement-independent inference rules. Each rule is sound with respect to the operational semantics of the language under consideration.

2.6 Modeling Methodology

A modeling methodology guides a modeler in constructing and validating models. A model is constructed using different primitives and constructs of the model specification language. The useful forms of

the constructs of a language for modeling can be obtained by either simplifying (or restricting) some complex constructs or by making new constructs using simpler ones. For example, in the ROBS (Rules and Object based Simulation) model specification language [Radiya and Sargent 1987], rules can be considered as special kind of objects (in its syntax). However, rules and objects have quite different meaning for modeling. A modeler analyses and specifies a representation of a problem entity by keeping the meanings of the constructs of the language. Thus the meaning of constructs available for modeling determines the modeling methodology. The new modeling methdology proposed for DES should allow behavioral patterns to be used in synchronizing the components of a model. The behavioral expressions, considered in section 2.3 and 3.3, may help to simplify the task of achieving synchronization and specifying validation conditions for complex models.

A model must be validated for its intended purpose. A validation methodology for a new formalism should have provosions for validating both model structure and its behavior.The behavioral validity should consist of operational validity as well as proving the other properties of model behavior using a proof system.

3. The Proposed Formalism

A particular formalism is now proposed for discrete event simulation. The elements of the proposed formalism are defined and discussed next.

3.1 The Model Specification Language

The language of the proposed formalism is based on the ROBS model specification language. As mentioned in section 2.1, a language may contain several less expressive languages. It has been shown in Radiya and Sargent (1988) that the ROBS model specification language can be used to specify models in event scheduling, activity scanning, and process interaction world views of DES besides the ROBS world view. Thus, the ROBS model specification language can be used to specify an interesting class of DES programming languages. However, synchronization expressions in ROBS are dependent only on the variables of a model and on message passing. The modeling methodology of ROBS can be enhanced by allowing complex behavioral patterns in synchronization expressions. The language of the formalism will include appropriate extensions of ROBS such that some of the behavioral expressions can be used as synchronization expressions. Some of the semantic concepts required to give precise meaning to the primitives and constructs of the model specification language are discussed in the next subsection.

3.2 Semantics

To specify the semantics of a language one considers only an "abstract" form of the syntax of a language. The abstracted syntax avoids some semantically irrelevant complications such as operator precedence. The semantics for a formal language is specified by defining the mappings from the primitives and constructs of the abstract syntax to the elements of the framework of the semantic concepts. In this paper, only the framework of the necessary semantic concepts is considered. It is shown how the meaning of a model can be given in such a framework. First, an informal description of the semantic concepts of the framework is presented. Such a framework consists of a model state set, the indexing element T, a model, and a select function. The concept of a model state is defined using the concept of a variable state, synchronization expressions, and agents. Then, it is shown how a select function is used to generate model behavior of a given model.

First, the concept of *variable state set* (Σ_V) is central to model specification languages (as well as to general purpose sequential programming languages) and the process of modeling. A variable state (σ_V) defines the current values of the variable identifiers in a model. The variable state set is a set of all functions (states) from variable identifiers to values. The concept of variable state is at the heart of the computer simulation languages based on traditional world views.

Second, the concept of *agent* is required. An agent is a function from model state set (Σ) to model state set (defined later). The set of all agents is denoted by **Agent**. Usually, an agent is a complex function which is a composition of several other functions (simpler agents) from model state set to model state set. In programming languages, agents are represented by atomic actions (commands, statements or instructions), constructs, programs, and subroutines. In DES programming languages based on the three basic world views, an agent is denoted by an event routine in event scheduling; an activity routine in activity scanning; and a process routine in process interaction world views.

Third, the concept of *synchronization expressions* is required. A synchronization expression is an expression containing predicates defined on Σ (i.e., Σ_V and Σ_S) and the indexing element T. Usually the domain of the indexing element T is $\mathbf{Real}^+$, where $\mathbf{Real}^+$ denotes the positive real numbers. A synchronization expression is a function from ($T \times \Sigma$) to {*true, false*}. Synchronization expressions are used to determine the order in which different agents are executed to produce the desired behavior of a model. The synchronization expressions needed to give semantics for most of the DES programming languages are the same as the boolean expressions in the respective languages. However, synchronization expressions are used very differently than boolean expressions, i.e., they have different semantics. Also, synchronization expressions as defined here contain predicates on Σ_V and Σ_S, so they include an expression like '**schedule** an agent **when** event A and event B occur together with event C or D'. The set of all synchronization expressions is denoted by **Sexp**.

Now, a *model state* (σ) can be defined. A model state σ consists of a variable state σ_V and a set (called σ_S) of pairs of the form, (ω, α) - where $\omega \in$ **Sexp** and $\alpha \in$ **Agent**. The model state set (Σ) is a set of all model states.

A model consists of an initialization agent to give initial value to model state, one or more agents, and a terminal agent to specify the termination condition for model execution. A model behavior is generated by executing the following *computation cycle*: select an agent from a given model state (begin with the initial model state as defined by the initialization agent) using a select function (ς); apply the selected agent (a function from Σ to Σ) to the given model state to obtain a new model state; and repeat the cycle.

3.2.1 Definitions

The precise definitions of the above informal description are stated next.

Variable State Set

Let **var** and **val** be the disjoint sets of symbols representing the sets of variables and values which variables may take. Then, *variable state* is σ_V : **var** $\rightarrow$ **val** and $\Sigma_V = \{\sigma_V\}$ = *variable state set.*

Synchronization Expressions

Let **Sexp** = {ω} be the set of all synchronization expressions defined using boolean connectives (such as **and**, **or**, etc) and predicates on Σ and the indexing element T. A synchronization expression ω ∈ **Sexp** denotes a function, ω : (T x Σ) → {*true, false*}. A partial function τ: **Sexp** → $\mathbf{Real}^+$ is defined for ω ∈ **Sexp** if ω contains a predicate defined on the T, i.e., τ returns the scheduled time associated with ω, if any.

Agents

Let **Agent** be the set of all agents. Then, **Agent** = {α | α : Σ → Σ} = The set of all functions from Σ to Σ. The agents are classified either as atomic actions or constructs. The atomic actions transforms the model state indivisibly, whereas constructs are defined in terms of one or more simpler agents, synchronization expressions, and other expressions like boolean expressions.

Model State Set

Now, the model state set can be defined using the definitions of Σ_v, **Sexp**, and **Agent**. *Model state set* = Σ = {σ} = $(\Sigma_v \times \Sigma_s)$ = {(σ_v, σ_s)}, where Σ_s = {σ_s} and $\sigma_s \subseteq$ (**Sexp** × **Agent**).

A model consists of an initialization agent (α_0), one or more agents, and a terminal agent. The initial model state is obtained by applying α_0 to φ (null model state), i.e., $\alpha_0(\phi) = \sigma^0$. Then, the next agent is selected by applying the select function to the current model state. The model execution is terminated when the terminal agent is executed. Here, the most general definition of the select function is considered. The only assumption made is that the select function never returns an agent whose scheduled time, if defined, is greater than scheduled time of any other scheduled agent and an agent with scheduled time greater than the current time is not selected until all the agents whose ω's are satisfied have been scheduled. This introduces nondeterminism into the formalism. In different programming languages further restrictions are made to reduce (often completely) nondeterminism. The selection function, ς : (T × Σ) → (T × **Agent**) is a partial function that returns the next selected agent and the scheduled time associated with it.

Then, the *computation cycle* which generates the model behavior is as follows.

$t \leftarrow t_0$
$\sigma \leftarrow \sigma_0$
repeat until *termination*
$t \leftarrow pr_1(\varsigma(\sigma))$
$\sigma \leftarrow pr_2(\varsigma(\sigma))\ (\sigma)$

where pr_i is a projection function, which gives ith element of a tuple.

3.3 Model behavior and validation

Traditionally, the behavior of a model (and a problem entity) is specified using the trajectories of some of the variables of a model. Here, the behavior of a model is defined by considering the trajectory (defined by relation →) of model state. Execution of a model results in the sequence of model states and atomic actions

$$\sigma^0 \xrightarrow{\alpha_1} \sigma^1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_i} \sigma^i \xrightarrow{\alpha_{i+1}} \cdots$$

where the σ^i's denote model states, the α_i's denote atomic actions, and the sequence $\alpha_1\alpha_2\ldots$ is the sequence of atomic actions resulting from the execution of agents specified in a model. The set of possible behaviors, {β}, is defined using the model state trajectories as follows.

1) $\beta = \Pi(b_1 \rightarrow b_2 \rightarrow \cdots \rightarrow b_m)$,
where the behavior β is said to have occurred during the model execution if the relation Π is satisfied by b_i's, where $b_i = f(\sigma)$ and f is an arbitrary function on σ. The relation → holds on b_i's if the relation → holds on the corresponding model states.

2) $\beta = \Pi(\beta_1, \ldots, \beta_n)$,
where the behavior β is said to have occurred if the behaviors β_1, ..., β_n satisfies the relation Π.

The above definitions are illustrated with the following examples. The variable trajectory of a variable, say x, that satisfies a required property can be considered as a specification of a model behavior. Namely, the model behaves in such a way that the variable trajectory of x satisfies the required property. This is expressed as $\beta = \Pi(b_1 \rightarrow b_2 \rightarrow \cdots \rightarrow b_m)$, where $b = f(\sigma)$, f is a function that returns the value of the variable x, and Π stands for the required property. The observation of the model's behavior of the form 'if events A and B occur together with events C or D then event G occurs' can also be specified using the above definitions as b is defined using a function on model state σ. The other useful behaviors can be defined using definition 2 as shown in the following example. Let β_1 and β_2 be the particular behaviors of the type activity, which are specified by $(b_1 \rightarrow b_2)$, where the states corresponding to b_1 and b_2 mark the beginning and the end of the activity. Let ‖ be a relation on two activities. The ‖ is true if only if its operand activities overlap. If $(\beta_1 \parallel \beta_2)$, then the activities β_1 and β_2 overlapped during the model execution.

The above generalized definition of model behavior can be used to specify generalized behavioral validity. One can express validity conditions like 'a model is valid only if $(\beta_1 \parallel \beta_2)$ is false for all model executions', i.e., β_1 never overlaps β_2 ('mutual exclusion' in concurrent programming terminology). Other validity conditions like 'a model is valid only if a variable trajectory of x satisfies certain statistical properties' can also be expressed in the proposed formalism.

4. Summary

The elements of and issues relating to defining a new formalism for DES have been identified and discussed in the paper. Some of the elements are defined and discussed in greater detail than others. To specify a complete formalism one needs to develop a theoretical framework appropriate for the analysis and specification of the different elements of a formalism. A language that allows behavioral expressions of the kind discussed in the paper for synchronizing different components (agents) of a model should be defined. The framework of semantic concepts presented in section 3.2 (with some possible extensions) can be used to give the precise mathematical meaning to the abstract syntax of the proposed language. A specification language, similar to Hoare's language of assertions, can be developed to specify the higher level behaviors and validity conditions for a model. A suitable proof system must be designed to prove the required behavioral properties of a model from its validity specification.

References

Backus, J.W. (1959). The Syntax and Semantics of the Proposed International Algebraic Language of Zurich ACM-GAMM Conference, *Proceedings of the International Conference on Information Processing*, June 1959, 125-132, UNESCO, Paris.

Hoare, C.A.R. (1969). An Axiomatic Basis for Computer Programming, *Communications of the ACM*, 12, 10 (October), 576-583.

Radiya, A. and R.G. Sargent. (1988). ROBS - Rules and Objects based Simulation, forthcoming in *Modelling and Simulation Methodology: Knowledge System Paradigms*, Oren, T.I., Zeigler, B.P., Elzas, M.S. (eds), North-Holland.

Sargent, R.G. (1987). An Overview of Verification and Validation of Simulation Models, *Proceedings of the 1987 Winter Simulation Conference.*

Tennent, R.D. (1976). The denotational Semantics of Programming Languages, *Communications of ACM* 19, 8 (August), 437-453.

Zeigler, B.P. (1976). *Theory of Modelling and Simulation*, John Wiley & Sons.

AUTHORS' BIOGRAPHIES

ASHVIN RADIYA is a Ph.D. student in the School of Computer and Information Science at Syracuse University. He holds a Bachelor's degree in Mechanical Engineering from Indian Institute of Technology - Bombay, India. His current research interests includes simulation modeling concepts, concurrent programming, and logic programming.

Ashvin Radiya
441 Link Hall
Syracuse University
Syracuse, NY 13244-1240
(315) 423-2820

ROBERT G. SARGENT is a professor of Industrial Engineering and Operations Research and a member of the Computer an Information Science faculty at Syracuse University. Dr. Sargent has served the Winter Simulation Conferences in several capacities, including being a member of the Board of Directors for ten years, Board Chairman for two years, General Chairman of the 1977 Conference, and Co-editor of the 1976 and 1977 Conferences *Proceedings.* Professor Sargent was Department Editor of Simulation Modeling and Statistical Computing for the *Communications of the ACM* for five years, has served as Chairman of the TIMS College on Simulation and Gaming, has been a member of the Executive Committee of the IEEE Computer Society Technical Committee on Simulation, and has receive service awards from ACM, IIE, and the Winter Simulation Conference Board of Directors. He Currently is an ACM National Lecturer and a Director-at-large of the Society for Computer Simulation. Dr. Sargent received his education at the University of Michigan. His current research interests include model validation, simulation methodology, simulation applications, performance evaluation, and applied operations research. Professor Sargent is a member of AIIM, New York Academy of Sciences, Sigma Xi, ACM, IIE ORSA, SCS, and TIMS and is listed in Who's Who in America.

Professor Robert G. Sargent
431 Link Hall
Syracuse University
Syracuse, NY 13244-1240
(315) 423-4348