Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

A COMPARISON OF SELECTED CONCEPTUAL FRAMEWORKS FOR SIMULATION MODELING

E. Joseph Derrick
Osman Balci
Richard E. Nance
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061, U.S.A.

ABSTRACT

The purpose of this paper is to compare thirteen Conceptual Frameworks (CFs) selected from among several categories of applicability to discrete-event simulation modeling. Each CF is briefly reviewed to provide the background information required for the comparison. Based on the insights gained in applying the CFs to the modeling of a complex traffic intersection system, the CFs are compared relative to their distinct characteristics and capabilities. Comparative comments are grouped according to the design guidance and implementation guidance features of the CFs. Conclusions highlight the inadequacies of the CFs and the importance of research in CF development.

1. INTRODUCTION

A fundamental human limitation, called the *Hrair Limit*, indicates that a human being cannot handle more than 7 ± 2 entities *simultaneously* (Miller 1956). Although this has been known for a long time, its implications for simulation model development have not been fully recognized. Simulation is used mostly for stochastic systems containing many simultaneous activities. The modeler needs conceptual assistance in representing complex systems and in successfully analyzing simulation models to effect system improvements.

The essential assistance is expected to be provided by the simulation model development environment (SMDE) currently being constructed following the automation-based software paradigm (Balci and Nance 1987a, 1987b). The conceptual guidance to be provided within the SMDE is currently being investigated. As part of this research, a comparison of thirteen conceptual frameworks is reported in this paper.

A *Conceptual Framework* (CF) is an underlying structure and organization of ideas which constitute the outline and basic frame that guide a modeler in representing a system in the form of a model. "Simulation strategy," "world view," and "formalism" are other terms used in lieu of CF.

The purpose of this paper is to compare the CFs selected from among several categories of applicability to discrete-event simulation modeling. Each CF is briefly reviewed in Section 2. Section 3 presents a comparison of the CFs based on their distinct characteristics and capabilities. Conclusions in Section 4 highlight inadequacies of the CFs and the importance of research in CF development.

2. REVIEW OF FRAMEWORKS

A brief review is presented in this section to provide the necessary background knowledge for the comparison of CFs in Section 3. The several references provide broader or more in-depth coverage.

2.1 The Classical Conceptual Frameworks

Popularized by a wide variety of Simulation Programming Languages (SPLs), Event Scheduling (ES), Process Interaction (PI), and Transaction Flow (TF) are perhaps the most well-known CFs used for simulation model implementation. Less predominant in the U.S. but well-accepted and recognized in the U.K., are Activity Scanning (AS) and its extension, the Three-Phase Approach (TPA). Balci (1988) employs a comparative exercise to bring out details in the differences among these CFs.

Event Scheduling. When using ES, the modeler considers the system of interest to be described in terms of events. Each identifiable event is associated with an *event routine* which is "a set of actions that may follow from a state change in the system" (Pidd 1984). This approach specifies that "some event is to take place at a determined time in the future" and can be scheduled (Kiviat 1969). Events that occur at a known future time are *determined events* (Nance 1981b). The scheduling of event routines is managed during implementation by the maintenance of the *event list*, a list of event notices or records which are ordered by time. Event routines contain the creation and destruction of event records (e.g., arrivals, departures, and other determined events), the scheduling of determined events (e.g., the bootstrapping of arrivals), and logical checks for *contingent events* (Nance 1981b). The occurrence of contingent events (sometimes called conditional events (Fishman 1973)) depends upon the satisfaction of some set of conditions which cannot be predicted in advance. The explicit scheduling of events results in an efficient model execution for a large class of models. Including checks for contingent events reduces the number of event records for processing and also improves efficiency. Yet, as Kreutzer (1986) points out, the model logic becomes fragmented (with the scattering of scheduling commands and the insertion of checks for contingent events) as the number of event routines and their potential interactions increase. Fragmentation makes the implementation less readable, less understandable, and harder to debug.

Activity Scanning. AS, prominent in the U.K., requires that the modeler identify the various types of objects in the system to be modeled, the activities which the objects perform, and the conditions under which these activities take place. Beyond Pidd (1984) and Kreutzer (1986), good descriptions are accessible (Fishman 1973; Zeigler 1976; Hooper 1986a, 1986b). AS uses a test set of boolean conditions, or *testhead* (Pidd 1984) to enable the determination of the state change that can initiate an activity. The testheads serve to link the various activities together and to produce the state transitions of the model objects and the interactions among them. In this way, the model is made up of modules or activity descriptions (testheads and associated resulting actions which await execution at the appropriate time.

Implementations of AS include a two-phase monitor or executive which performs a *time scan* (to ascertain the time increment or update to the system clock) and an *activity scan* (a check of all testheads to determine which of the activities are to be next executed (Pidd 1984)). In its purest implementation, AS uses the fixed-time increment time flow mechanism; the activity scan of testheads is strictly a scan of state conditions (Kiviat 1969; Balci 1988). This state-based approach is sometimes mixed with a time-

based approach to increase implementation efficiency by attaching *time cells* to model objects or activities (Pidd 1984; Kreutzer 1986). The time cells hold activity occurrence times and enable the use of the variable-time increment method for updating the system clock.

The Three-Phase Approach. The Three-Phase Approach (TPA) is a modification of AS which improves execution efficiency. The approach categorizes activities as B-activities or C-activities (Tocher and Owen 1961; Tocher 1963). The B-activities are the bound-to-occur or book-keeping activities that represent the unconditional state changes (unconditional events) which can be scheduled in advance. The C-activities are the conditional or co-operative activities that represent the state changes which are conditional upon the co-operation of different objects or the satisfaction of specific (compound) conditions. Because B-activities become due at a determined time, their testheads may be dropped and they can be scheduled using an events list technique (O'Keefe 1986). Since the traditional, repetitive activity scan and testhead checks do not need to be done on the B-activities, unnecessary scans are removed and efficiency is improved. C-activities with testheads must enter the usual, repetitive activity scan of their testheads. Comprehensive descriptions of the TPA are available in current literature (Crookes 1982; Pidd 1984; O'Keefe 1986; Balci 1988).

Process Interaction. Instead of the event or activity, PI (Kiviat 1969; Fishman 1973; Pidd 1984) uses the process as its basic building block. The process represents a sequence of events and interspersed activities through which a specific object moves. As the object moves through its process, it may experience certain delays and be blocked in its movement. Delays can be time-based and determined (e.g., service times, arrival times) or state-based and contingent (e.g., wait-until situations). Objects experience periods of activity during process execution and periods of inactivity or delay. When an object experiences a delay in its process and becomes "passive," another model object is allowed to become "active" (Franta 1977) initiating or resuming its process. Such delays are incurred and execution is shifted (to another object) at *interaction points* (Kiviat 1969). An object process returns to an active state following such interaction at its *reactivation points*. PI enables the modeler to clearly grasp a model's structure since each object or class of objects can be represented by a single, coherent process rather than through multiple event routines (Kiviat 1969; Fishman 1973).

Transaction Flow. TF handles the time and state relationships of the model in exactly the same manner as the PI CF. Several different and distinct characteristics are noted. "Transactions" are created and moved through the blocks, executing specialized actions that are "associated" with each block. The block structure generates a rigid structure which limits the "examination and communication" among system components (Shub 1980). In addition, as objects (transactions) pass through these blocks, "predefined processes" are activated which are hidden to the modeler. Statement languages with their lower level primitives provide generality and flexibility to the modeler. Tocher (1965) characterizes SPLs as machine or material-oriented. He further defines transactions to be material or temporary objects. In a machine-oriented view, servers (machines) are the dominating and active influence in the model (Kreutzer 1986). They obtain the material objects (transactions), operate on them, and place them in (or remove them from) queues. TF promotes material-oriented models in which the transactions are the dominant objects. Servers, now passive, are "acquired, held, and released again" by the transactions which flow from machine to machine (Kreutzer 1986).

2.2 Other Discrete-Event Conceptual Frameworks

Conceptual frameworks offering direct application to the field of discrete-event simulation and modeling have emerged within the last decade. These frameworks are the System Theoretic Approach (STA), the Conical Methodology (CM), and the Condition Specification (CS).

System Theoretic Approach. Under the STA (Zeigler 1976, 1984), a modeler can identify the static and dynamic structure of the model. The STA, based upon set theory and the systems modeling formalism, provides a comprehensive, yet general, model representation and allows hierarchical decomposition and abstraction. A system model can be informally represented by describing its:

- components - "the parts from which the model is constructed,"
- descriptive variables - "tools to describe the conditions of the components at points in time," and
- component interactions - "the rules by which components exert influence on each other, altering their conditions and so determining the evolution of the model's behavior over time."

The Discrete Event System Specification (DEVS) (Zeigler 1976) is a formal specification for discrete-event models which incorporates these concepts and provides for a variable-time increment time flow mechanism. DEVS provides the static structure of the model. In addition, model dynamic structure is obtained via the rules of component interaction. DEVS is developed from a more general formalism, the Systems Modeling Formalism (Zeigler 1984), which also contains a continuous time base. Although the formalisms of the STA make its direct use cumbersome, recent work surrounding the development of a PC-based environment (PC-Scheme) is helping to improve the modeler's ability to build model specifications based upon DEVS and the STA (Zeigler 1987).

The Conical Methodology. The CM (Nance 1981a, 1987) divides model representation tasks into two stages: model definition and model specification. The CM is based on the object-oriented paradigm and seeks to achieve five primary objectives (Nance 1987): model correctness, testability, adaptability, reusability, and maintainability. CM mandates a strict separation between model representation and model execution. *Top-down definition* and *bottom-up specification* techniques are at the core of its procedural guidance. Top-down model definition produces a static model representation and is accomplished through a hierarchical decomposition of the model into successive submodels. At each level of decomposition, attributes, including attribute dimensionality and range of values, are assigned (to the particular submodel associated with that level) and are classified by type in accordance with Nance's (1987) taxonomy tree. Bottom-up specification produces a model representation which contains the necessary information for model dynamics. Specification, "...the process of describing system behavior so as to assist the system designer in clarifying his conceptual view of the system" (Barger 1986), is started at some base-level submodel in the decomposition hierarchy and is performed at successively higher levels until the model level is reached. The time flow mechanism to be used in building the model is not dictated.

The Condition Specification. CS produces a model specification that can be analyzed to: "detect potential problems with the specification," "assist in the construction of an executable representation of the model," and "construct useful model documentation" (Overstreet and Nance 1985). The CS, attributed to Overstreet (1982), is the principal target form for bottom-up specification within the CM. The CS provides a representational foundation upon which additional analysis can be conducted for efficient model implementation (Overstreet and Nance 1985; Nance and Overstreet 1987). The analytic and diagnostic capabilities of the CS are an extremely desirable and significant strength. The principal components of the CS are the *interface specification, the specification of model dynamics*, and the *report specification* (Overstreet and Nance 1985). The input and output attributes of the model are described within the interface specification. The specification of model dynamics consists of a set of *object specifications* and a *transition specification*. The object specification is a complete listing of all objects and their attributes. A value range

is given for each attribute. The transition specification contains the description of model dynamics in the form of condition and action pairs (CAPs). The report specification is defined for the data output or model results.

2.3 More Generic Conceptual Frameworks

The following conceptual frameworks have applications within and beyond the domain of discrete-event simulation and modeling. The first two presented in this section, the Entity-Relationship-Attribute (ERA) and the Entity-Attribute-Set (EAS) frameworks, have their conceptual underpinnings from the field of database design. Yet, ERA is closely akin to EAS which has roots in the popular SPL, SIMSCRIPT (CACI 1983; Kiviat, et al. 1983). Finally, the Object-Oriented Paradigm (OOP) plays a significant role within software engineering and programming methodology and finds its origins in the early development of SIMULA.

Entity-Relationship-Attribute. ERA is based upon the Entity-Relationship (ER) model of Chen (1976). The ER model is a data model which is rooted in set and relation theory. Chen introduces the ER model through the context of levels of logical views of data, and develops the ER model for two of these levels: the *conceptual* level ("information concerning entities and relationships which exist in our minds" and the *representational* level (an information structure or organization of information in which data represents entities and the relationships which exist among them). We are concerned with the conceptual view of the model (of entities, their relationships, and their values) as it exists in our minds at the first level. Entities may be grouped into *entity sets*. Relationships may also specify similar groupings called *relationship sets*. Entity and relationship information is maintained within sets which contain values (called *value sets*). Entities and values are linked to one another by attributes. In addition to entities, relationships can also have attributes. Successful application of ERA relies on the proper identification of set membership and semantics, definition of value sets and attributes, and organization of data into relations.

Entity-Attribute-Set. The terms *entity, attribute*, and *set* are concepts central to EAS (Markowitz, et al. 1983):

- entity - "some concrete or abstract 'thing' represented by the simulation,"
- attribute - "some property or characteristic of the entity," and
- set - "an ordered collection of entities."

In EAS, entities of interest in the system are maintained in a database. The database also maintains complete information on a particular entity to include attribute and value data, the identification of the sets to which the entity belongs or which it owns, and the membership of the sets which it owns. A key characteristic of EAS is that sets are *ordered*. This ordering may be strictly on a first-in, first-out (FIFO) basis or in some other determined order. Hierarchical decompositions and tree-like structures of the system are easily defined. Set ordering provides the ability to represent timed events and a system state can be represented in a database. A function can be determined which transforms the database from state to state. Model dynamics are difficult to represent.

The Object-Oriented Paradigm. The OOP is viewed as a framework for system design. According to Meyer (1987), "object-oriented design may be defined as a technique which, unlike classical (functional) design, bases the modular decomposition of a software system on the classes of objects the system manipulates, not on the functions the system performs." Functions tend to change in order to adapt to changing needs whereas objects remain more or less constant (Meyer 1987). The paradigm is principally characterized by two features: (1) *encapsulation* of data and operations and (2) an *inheritance* mechanism for developing object hierarchies. An object can be considered to be "encapsulated", an "armor-plated" entity (Cox 1986) with "private data and a set of operations that can access that data." The use of objects therefore improves the reliability and maintainability of system code. Additionally, by inherent abstraction, the object improves the view of the system by introducing a higher level perspective and promotes reusability of code. The principles of *modularity, abstract data typing*, and *information hiding* are accommodated. Inheritance is the ability to define new objects from existing objects by extending, reducing, or otherwise changing their functionality. New instances of an object class can be easily created which automatically inherit the attributes of that class definition. Inheritance supports hierarchical structures that are commonly found in the real world and provides substantial benefit to the user by improving his understanding and view of the system.

2.4 Frameworks with Applications Potential

We conclude our review of conceptual frameworks with coverage of Structured Modeling (SM) and the Process Graph Method (PGM). Both frameworks, active in other fields, show promise for applications to discrete-event simulation model development.

Structured Modeling. SM (Geoffrion 1987, 1989) seeks to provide not only a generic *framework* for model representation but also an *environment* to meet total model developmental needs throughout the model life-cycle. SM can be used in a top-down model design strategy that embodies a stepwise refinement approach and which results in a well documented, easily communicated design. SM aims to be broadly applicable and technique independent (mathematical programming, database theory for data models, conceptual graphs and knowledge representation). The SM framework for model representation uses "a hierarchically organized, partitioned, and attributed acyclic graph "for model semantic and mathematical structure. The framework can be decomposed into **elemental, generic**, and **modular structures.**

Model *elemental structure* is generated through the formation of model elements of five types (primitive and compound entity, attribute, function, and test) into a directed graph in which the nodes represent the elements. The construction of an elemental structure (Geoffrion 1987) is intended to completely capture "the definitional detail of a specific model instance." Table representations of the elemental structure, *elemental detail tables*, contain instance data and low-level model information which is necessary for a complete model specification.

The *generic structure* accomplishes the grouping of elements according to "natural familial" boundaries. The generic structure thus provides the modeler with a natural view of the system under study. The *modular structure* (Geoffrion 1987) is a further refinement on the generic structure. The modular structure is created in order to bring into play the concepts of data abstraction and information hiding. "Modules" are formed by grouping the genera "into conceptual units ... according to commonality or semantic relatedness." Modules, themselves, can then be grouped into higher order modules. In this way, complex models can be simplified into a representation which will be better understood. A modeling language, SML, supports the central concepts of SM.

Process Graph Method. The PGM is derived from the parallel computation model which was suggested by Karp and Miller (1966) and later improved upon by the U. S. Navy (Kaplan 1987). PGM is used primarily for the development of signal processing models. The basis for the PGM is the process graph, a directed graph of *nodes* and *arcs* which is classified as a *data flow* model. Three of the primary benefits of the process graph are its parallel computation capabilities for greater throughput, the ease at which modelers can perform top-down design, and portability of applications. Each node in the process graph represents a *primitive function* (some type of computation or process) or may alternatively represent a *subgraph* which is itself a process graph. Such a convention allows the modeler to use abstraction and modularity to represent complex models in a fashion that is more easily understood. Arcs represent queues which contain the input and

output data needed by nodal primitive functions for execution. Nodes execute only when the data necessary for execution are available at the input queues. Such a node execution scheme allows multiple computations to be performed in parallel thus generating greater throughput.

3. COMPARISON OF FRAMEWORKS

In this section, the selected CFs are compared on the basis of their application to a real-world traffic intersection (TI) modeling problem (Derrick 1988). The TI is selected because it offers complexity of model component interaction unlike that found in the usual textbook examples. The TI contains a single traffic light with north, south, east, and west directions which controls vehicular traffic in each of the intersection's eleven lanes. The central intersection space is conceptually divided into thirty-five blocks through which the vehicles travel. The blocks in a vehicle's path are used as locators for that vehicle as it moves through the intersection and enable the representation of a smoothly flowing traffic pattern. Care is taken to be circumspect in drawing conclusions from these applications under such a restricted domain. The intent, however, is to develop a starting point for discussion by accomplishing a thorough investigation under a single problem domain.

Two basic types of guidance can be provided by the CFs in constructing model representations. First is implementation guidance (algorithmic, managerial, supervisory) which directly impacts the subsequent executable form of any model representation. Secondly, CFs can provide **design** (structural, existential, skeletal) guidance. Here, the modeler is aided in his definition of the model's *static* and *dynamic* structure as he identifies the objects (components, entities), their attributes, and their rules of interaction. We explore the comparisons of the CFs with regards to the types of guidance that each explicitly makes available to the modeler. Grouping our comparisons by guidance type (implementation or design) improves the clarity and meaningfulness of our comments. We caution the reader to avoid generalizing the comparisons to all problem domains.

3.1 Implementation Comparisons

The classical CFs provide implementation level guidance to the modeler. This guidance may be considered to assist the modeler in the *mode of sequencing* and in the *method of sequencing*. The mode of sequencing reflects the world view or *Weltansicht* (Lackner 1962), the view promoted by the CF that effects model transformation from state to state. Viewing the model as being composed of events, activities, or processes influences the programming task and determines the coding format (event routines, activity descriptions, or process descriptions). The method of sequencing is the guidance perspective relating to the algorithmic nature of the CF, e.g., whether by explicit scheduling of events, scanning of conditions, or by concurrent control of object interactions.

Comparing Mode of Sequencing. Within event routines, we note that the burden is placed on the modeler to include all conditional testing (on conditions other than time). As the complexity of model interaction increases, the modeler is less able to accurately make such determinations and maintain consistency in the model. The programmed model becomes error-prone, hard to modify, and difficult to debug. In addition, the model logic surrounding the occurrence of a particular event can become fragmented and scattered throughout the code, making the code hard to read and understand.

Activity descriptions free the modeler from having to explicitly specify the interactions and relations among events (Laski 1965; Kreutzer 1986). Under AS or the TPA, the programmed model is readable and simple, primarily due to the clarity achieved through the grouping of the conditional tests (Kiviat 1969; Kreutzer 1986).

The ability to apply an object-oriented approach and confine all information pertinent to a single process description improves readability and understanding, naturally enhances maintainability, and reduces problems in debugging. PI requires that a modeler signal an object's activation and passivation and explicitly control the queueing and competition for resources. However, TF automatically accomplishes many of these tasks within its block structure. Support of the modularity principle, afforded by both the activity-oriented (grouping by state conditions) and the process-oriented (grouping by object process) frameworks, speed application development.

Comparing Method of Sequencing. Under ES, the direct determination of event selection, execution, and clock update through the use of the events list produces an efficient execution when the model is composed of less interactive, more independent objects (Kiviat 1969; Nance 1971; Birtwistle, et al. 1985; Hooper 1986b). Under the same conditions, AS produces a less efficient representation for execution since with an increase in the number of independent components, the number of repetitive, redundant, and unnecessary scans also increases. However, whenever a model is characterized by a large number of primarily dependent and interactive components, AS demonstrates improved execution efficiency (Nance 1971).

The TPA improves execution efficiency while retaining the advantages of AS. With the use of reactivation points and ability to effect concurrency among objects, the process-oriented frameworks (PI and TF) are able to simplify a programmed model. They are preferred when the model is composed of a balance of independent and dependent components (Hooper 1986b).

Table 1 summarizes the comparative features just discussed, giving a panorama of the key characteristics of the classical CFs when used to build models like the TI which have many components and component interactions. The ES, AS, TPA, PI, and TF CFs provide a wide range of implementation guidance characteristics, compared and discussed above.

3.2 Design Comparisons

CFs other than the classical CFs provide design guidance. We compare them relative to their ability to effectively assist the modeler in his design of the *static* and *dynamic* structure of the model. Additionally, the identification of relationships (how the objects are bonded or related to one another) is also of concern.

Model Static Structure. The CFs discussed here provide limited guidance for the *identification of objects and their attributes* and coerce the modeler to perform this identification task.

- OOP - The OOP conceptually stipulates that all information for a given object (including attribute information) is encapsulated within that object's description.
- PGM - Object and attribute information is contained in variable attribute and queue attribute tables.
- ERA and EAS - Both ERA and EAS derive their conceptual basis from the objects (entities) and attributes that make up a given model.
- CM - CM, an extension of the OOP, very clearly guides the modeler in a top-down definition of model objects and attributes.
- SM - The elemental and generic structures enable a full designation of objects and their attributes.
- CS - The object specification includes provision for object and attribute identification and establishes the static structure of the model but focuses on the specification of model dynamics.

Table 1: Characteristics of Classical Conceptual Frameworks

CONCEPTUAL FRAMEWORK	ES	AS	TPA	PI	TF
CONDITIONS FOR MAXIMUM EFFICIENCY	Independent Objects	Dependent Objects	Independent or Dependent Objects with resource competition	Balance of Independent and Dependent Objects	Balance of Independent and Dependent Objects
BURDEN ON MODELER	High	Low	Low	Moderate#	Low
BURDEN ON EXECUTIVE	Low	High	High	High	High
MODEL LOGIC DESCRIPTION	Fragmented throughout event routines	Conditional logic concentrated at testheads	Conditional logic concentrated at testheads and determined logic concentrated at B-activities	Concentrated in modules of process descriptions	Concentrated in modules of block segments
MAINTAINABILITY	Low	High†	High†	High§	High§
NATURAL REPRESENTATION CAPABILITY	Poor	Good	Good	Excellent	Excellent
DEVELOPMENTAL TIME, EFFORT REQUIRED	Very high	Low	Low	High	Low
APPLICATION LINES OF CODE‡	1312	--	—	1778	443

† Due to localization of state with grouping of conditional testing
§ Due to localization of object and modularization of process descriptions
# Due to modeler responsibilities in activation, passivation, and queueing for resources
‡ Lines of source code for event routines or process descriptions; applicable to research applications [Derrick 1988] only; does not include code for initialization or statistics collection

- STA - Model components and descriptive variables relate object and attribute information.

Model Dynamic Structure. The relationships and rules of *dynamic design guidance*, once specified, provide the motive force for affecting the state changes among the model objects. Therefore, this aspect of guidance is critical to producing a working, accurate model representation. The CS, STA, and CM provide explicit support, albeit limited, for accomplishing this task. We note that the dynamic design guidance provided by these CFs is independent of mode of sequencing.

The transition specification of the CS guides a modeler for this purpose. The transition specification, however, only provides limited guidance in format and syntax to the modeler for developing dynamic relationships. The STA via the DEVS formalism also provides dynamic design guidance. The "necessary equipment" to specify model dynamics is available to the modeler in the form of the time advance and transition functions. Similar to the CS, the STA provides only limited guidance in format and syntax (notation). Set theoretic notation and the intricate details of the DEVS formalism makes"using the equipment" a difficult task for the modeler. Bottom-up specification of the CM enables the specification of model dynamics but is much less structured than the CS and STA. It has been conjectured (Geoffrion 1987) that SM can accommodate the dynamic relationships of discrete-event models but this applicability is yet to be shown.

Model Relationships. A hierarchical decomposition capability supports the definition of *1:1* or *1:m* relationships among objects. Hierarchical decompositions (from an object or entity viewpoint) are possible when the model is influenced by OOP, PGM, ERA, EAS, CM, SM, or STA CFs. The inheritance features of the OOP and PGM described in their applications allow hierarchical decompositions. The use of entity and relationship sets enable ERA to easily handle hierarchical decompositions; the entity-relationship diagram makes it easy to define the *1:m* relationships. In a similar manner, the use of sets in EAS makes such decomposition possible. Both CM (with its OOP orientation) and SM (with its hierarchically organized structures) provide the flexibility of hierarchical decompositions. The set orientation of the STA allows the establishment of object hierarchies. Since CS can be extended to include sets, hierarchical decompositions should be possible.

Although our experience from applications to the TI system is limited concerning abilities of the CFs for *m:n* relationships, we offer the following perceptions:

- ERA offers the most straightforward approach for *m:n* relationships when assisted by the entity-relationship diagram.
- CM and SM suggest excellent capabilities for *m:n* relationships based on the experience gained from the literature review and in performance of the TI applications. Ease in use of the SM for accomplishing this within the SML is currently limited.
- EAS, STA, and CS allow definition of *m:n* relationships but without the direct, natural clarity of the above approaches.
- OOP and PGM should permit designation of *m:n* relationships since an object may represent a set of objects and a process graph node may represent an underlying network of process graph nodes.

Table 2 summarizes the comparisons of CFs based on design guidance. Each CF that has been considered in this section provides a level of design guidance that is sufficient to adequately define model structure. Depending on the aspect, certain CFs maintain a clear advantage for the modeler. Table 3 lists the CFs according to the type of guidance they provide.

Table 2: Comparisons Based on Design Guidance

CONCEPTUAL FRAMEWORK	OOP	ERA	EAS	CM§	SM§	CS§	STA§	PGM§
OBJECT AND ATTRIBUTE NAMING	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
CAPABILITY FOR DYNAMIC DESIGN SPECIFICATIONS	No	No	No	Limited	No	Limited	Limited	No
CAPABILITY FOR *ONE-TO-MANY* RELATIONSHIPS	Yes	Yes	Yes	Yes	Yes	Yes‡	Yes	Yes
CAPABILITY FOR *MANY-TO-MANY* RELATIONSHIPS	Yes†	Excellent	Yes†	Excellent	Good	Limited	Limited	Yes†

† Not observed
‡ With set extension
§ Includes documenting features

Table 3: Types of Guidance Provided by the Conceptual Frameworks Under Review

IMPLEMENTATION	DESIGN (STATIC)	DESIGN (DYNAMIC)
ES	EAS	CM
AS	ERA	CS
TPA	CM	STA
PI	SM	
TF	OOP	
	PGM	
	CS	
	STA	

4. CONCLUSIONS

Thirteen selected CFs for discrete-event simulation modeling are briefly reviewed and compared. The review introduces the fundamentals of four groups of CFs: (1) the classical (execution-oriented) group, (2) emerging discrete-event CFs, (3) CFs with demonstrated viability to other than discrete problem domains, and (4) CFs exhibiting potential for discrete-event simulation modeling. The comparison contributes a detailed and instructive discussion of the characteristic differences among the CFs, based upon their individual application to a complex real-world traffic intersection system.

ACKNOWLEDGMENT

This research was sponsored in part by the Naval Sea Systems Command and the Office of Naval Research under contract N60921-83-G-A165 through the Systems Research Center at VPI & SU.

REFERENCES

Balci, O. (1988). The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages. In: *Proceedings of the 1988 Winter Simulation Conference* (M.A. Abrams, P.L. Haigh, and J.C. Comfort, eds.). Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 287-295.

Balci, O. and Nance, R.E (1987a). Simulation Model Development Environments: A Research Prototype. *Journal of the Operational Research Society 38,* 753-763.

Balci, O. and Nance, R.E. (1987b). Simulation Support: Prototyping the Automation-Based Paradigm. In: *Proceedings of the 1987 Winter Simulation Conference* (A. Thesen, H.Grant, and W.D. Kelton, eds.). Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 495-502.

Barger, L.F. (1986). The Model Generator: A Tool for Simulation Model Definition, Specification, and Documentation. Unpublished M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Birtwistle, G., Lomow, G., Unger, B., and Luker, P. (1985). Process Style Packages for Discrete Event Modeling: Experience from the Transaction, Activity, and Event Approaches, *Transactions of the Society for Computer Simulation 2*, 27-56.

Chen, P.P. (1976). The Entity-Relationship Model- Toward a Unified View of Data. *Association for Computing Machinery Transactions on Database Systems 1*, 9-36.

Cox, B.J. (1986). *Object-Oriented Programming: An Evolutionary Approach.* Addison-Wesley, Reading, Massachusetts.

Crookes, J.G. (1982). Simulation in 1981. *European Journal of Operational Research 9*, 1-7.

CACI, Inc. (1983). *SIMSCRIPT II.5 Reference Handbook.* CACI, Inc.-Federal, Los Angeles, California.

Derrick, E.J. (1988). Conceptual Frameworks for Discrete-Event Simulation Modeling. Unpublished M.S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Fishman, G.S. (1973). *Concepts and Methods in Discrete Event Digital Simulation.* John Wiley and Sons, New York.

Franta, W.R. (1977). *The Process View of Simulation.* North Holland, Amsterdam.

Geoffrion, A.M. (1987). An Introduction to Structured Modeling. *Management Science 33*, 547-588.

Geoffrion, A.M. (1989). The Formal Aspects of Structured Modeling. *Operations Research 37*, 30-51.

Hooper, J.W. (1986a). Activity Scanning and the Three-Phase Approach. *Simulation 47*, 210-211.

Hooper, J.W. (1986b). Strategy-Related Characteristics of Discrete-Event Languages and Models. *Simulation 46*, 153-159.

Kaplan, D.J. (1987). The Process Graph Method, An Iconic Method of Controlling Networks of Processors. In: *Proceedings of the 1987 Summer Computer Simulation Conference* (J.Q.B. Chou, ed.). The Society for Computer Simulation, San Diego, California, 219-227.

Karp, R.M. and Miller, R.E. (1966). Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing. *SIAM Journal on Applied Mathematics 14*, 1390-1411.

Kiviat, P.J. (1969). Digital Computer Simulation: Computer Programming Languages. Memorandum RM-5883-PR, The Rand Corporation, Santa Monica, California.

Kiviat, P.J., Markowitz, H., and Villanueva, R. (1983). *SIMSCRIPT II.5 Programming Language.* Revised. CACI, Inc.-Federal, Los Angeles,California.

Kreutzer, W. (1986). *System Simulation: Programming Styles and Languages.* Addison-Wesley, Reading, Massachusetts.

Lackner, M.R. (1962). Toward a General Simulation Capability. In: *Proceedings of the AFIPS 1962 Spring Joint Computer Conference.* National Press, Palo Alto, California, 1-14.

Laski, J.G. (1965). On Time Structure in Monte Carlo Simulations. *Operational Research Quarterly 16*, 329-339.

Markowitz, H.M., Malhotra, A., and Pazel, D.P. (1983). The ER and EAS Formalisms for System Modeling and the EAS-E Language. In: *Entity-Relationship Approach to Information Modeling and Analysis: Proceedings of the Second International Conference on Entity-Relationship Approach* (P.P. Chen, ed.). North Holland, Amsterdam, 29-48.

Meyer, B. (1987). Reusability: The Case for Object-Oriented Design. *Institute of Electrical and Electronics Engineers Software*, 50-64.

Miller, G.A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review 63*, 81-97.

Nance, R.E. (1971). On Time Flow Mechanisms for Discrete System Simulations. *Management Science 18*, 59-73.

Nance, R.E. (1981a). Model Representation in Discrete Event Simulation: The Conical Methodology. Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Nance, R.E. (1981b). The Time and State Relationships in Simulation Modeling. *Communications of the Association for Computing Machinery 24*, 173-179.

Nance, R.E. (1987). The Conical Methodology: A Framework for Simulation Model Development. In: *Proceedings of the Conference on Methodology and Validation* (O. Balci, ed.).The Society for Computer Simulation, San Diego, California, 38-43.

Nance, R.E. and Overstreet, C.M. (1987). Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications. *Transactions of the Society for Computer Simulation 4,* 33-55.

O'Keefe, R.M. (1986). The Three-Phase Approach: A Comment on `Strategy-related Characteristics of Discrete-event Languages and Models'. *Simulation 47,* 208-210.

Overstreet, C.M. (1982). Model Specification and Analysis for Discrete Event Simulation. Ph.D. Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Overstreet, C.M. and Nance, R.E. (1985). A Specification Language to Assist in Analysis of Discrete Event Simulation Models. *Communications of the Association for Computing Machinery 28,* 190-201.

Pidd, M. (1984). *Computer Simulation in Management Science.* John Wiley and Sons, New York.

Shub, C. M. (1980). Discrete Event Simulation Languages. In: *Proceedings of the 1980 Winter Simulation Conference 2* (T.I. Oren, C.M. Shub, and P.F. Roth, eds.). Institute of Electrical and Electronic Engineers, Piscataway, New Jersey, 107-124.

Tocher, K.D. (1963). *The Art of Simulation.* English Universities Press, London.

Tocher, K.D. (1965). Review of Simulation Languages. *Operational Research Quarterly 16,* 189-217.

Tocher, K.D. and Owen, D.G. (1961). The Automatic Programming of Simulations. In: *Proceedings of the Second International Conference on Operational Research,* (J. Banburg and J. Maitland, eds.). John Wiley and Sons, New York, 50-68.

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

Zeigler, B.P. (1984). System-Theoretic Representation of Simulation Models. *IIE Transactions 16,* 19-34.

Zeigler, B.P. (1987). Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment. *Simulation 49,* 219-229.

AUTHOR'S BIOGRAPHIES

RICHARD E. NANCE is a professor of Computer Science and the director of the Systems Research Center at Virginia Polytechnic Institute and State University. He received B.S. and M.S. degrees from N.C. State University in 1962 and 1966, and Ph.D. degree from Purdue University in 1968. He has served on the faculties of Southern Methodist University and Virginia Tech, where he was Department Head of Computer Science, 1973-1979. Professor Nance has held research appointments at the Naval Surface Weapons Center and at the Imperial College of Science and Technology (UK). Within ACM, he has chaired two special interest groups: Information Retrieval (SIGIR), 1970-71 and Simulation (SIGSIM), 1983-85. He has served as Chair of the External Activities Board, the Outstanding Service Awards Subcommittee, the ad hoc Conference Procedures Committee and the ad hoc Film Committee that produced "Computers In Your Life."

He currently serves on the Editorial Panels of *Communications of the ACM* for research contributions in simulation and statistical computing, and *Journal of Operations Research and Computer Science* for contributions in simulation. The author of papers on discrete event simulation, performance modeling and evaluation, and computer networks, Professor Nance has served as Area Editor for Computational Structures and Techniques of Operations Research 1978-82, and as Department Editor for Simulation, Automation and Information Systems of *IIE Transactions,* 1976-81.

Professor Richard E. Nance, Director
Systems Research Center
320 Femoyer Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061 U.S.A.
(703) 961-6144

OSMAN BALCI is an associate professor of Computer Science at Virginia Polytechnic Institute and State University. He received B.S. and M.S. degrees from Bogazici University (Istanbul, Turkey) in 1975 and 1977, and M.S. and Ph.D. degrees from Syracuse University (N.Y.) in 1978 and 1981. He is currently the simulation and modeling category editor of *ACM Computing Reviews*, the general chairman of the ORSA CSTS conference on the Interfaces of Computer Science and Operations Research, and the proceedings editor of the 1990 Winter Simulation Conference. He has served as a member of the organizing committee of the ORSA CSTS conference on Impact of Recent Computer Advances on Operations Research, the vice-chairman of ACM SIGSIM, the program chairman and proceedings editor of the SCS conference on Methodology and Validation, and an associate editor of *Simuletter.* He has been a consultant for Planning Research Corporation, VM Software Inc., and Central Intelligence Agency. His current research interests center on simulation model development environments, credibility assessment of simulation results, software engineering, and performance evaluation. Dr. Balci is a member of Alpha Pi Mu, Sigma Xi, ACM, IEEE CS, and ORSA.

Dr. Osman Balci
Department of Computer Science
562 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061 U.S.A.
(703) 231-4841

E. JOSEPH DERRICK received his B.S. degree in Electrical Engineering in 1974 from the U.S. Naval Academy. After several years of military and government service, Mr. Derrick continued his formal education at Virginia Polytechnic Institute and State University (VPI & SU) receiving the M.S. degree in Computer Science and Applications in 1988. Mr. Derrick is currently a Ph.D. student at VPI & SU. His research interests are simulation methodology, simulation model development environments, software engineering, and human/machine interaction. He is a student member of ACM, IEEE CS, and SCS.

Mr. Joseph Derrick
Department of Computer Science
335 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061 U.S.A.
(703) 231-7371