Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

THE HISTORY OF SIMULATION MODELING

Stephen D. Roberts

Edward P. Fitts Department of Industrial and Systems Engineering
North Carolina State University
111 Lampe Dr., Daniels Hall Room 400
Raleigh, NC 27695-7806, USA

Dennis Pegden

Simio LLC
504 Beaver Street
Sewickley, PA 15143, USA

ABSTRACT

During the past half-century simulation has advanced as a tool of choice for operational systems analysis. The advances in technology have stimulated new products and new environments without software standards or methodological commonality. Each new simulation language or product offers its own unique set of features and capabilities. Yet these simulation products are the evolution of research, development, and application. In this paper we interpret the historical development of simulation modeling. In our view simulation modeling is that part of the simulation problem-solving process that focuses on the development of the model. It is the interpretation of a real production (or service) problem in terms of a simulation language capable of performing a simulation of that real-world process. While "interpretation" is in the "eyes of the beholder" (namely us) there are some historical viewpoints and methods that influence the design of the simulation model.

1 INTRODUCTION

1.1 Models vs. Modeling

Formal mathematical and statistical models of operations have been developed over several decades of research. However, when a model is applied beyond its assumptions, it leaves open the question of its applicability and the modeler is faced with adapting the model or developing a new model. The adaption or creation of a new analytic model is a significant task and that challenges the analytical skills of the developer, and may require considerably more time and effort, perhaps without tangible results.

The value of a model is its use. Since individual models have limitations and their extension can be daunting, some researchers and practitioners prefer a more flexible modeling environment. Simulation has long been viewed as modeling tool that has a very broad development environment and does not require mathematical or statistical sophistication for model development and use. Furthermore, to ease the burden of the development of simulation models, a variety of simulation languages have been developed. Each simulation language offers its own modeling constructs within which a simulation model can be constructed, simulated, and analyzed. While the languages have changed over time, the modeling approach is inextricably linked to its simulation execution perspective.

1.2 Modeling

The purpose of this paper is to provide a personalized (and therefore limited) history of simulation modeling as it has been largely popularized through the Winter Simulation Conference. Using simulation in a problem-solving context requires several proficiencies:

- Defining the problem correctly and setting the objects.
- Using modeling concepts for abstracting the essential features of the system into a model.
- Collecting and compiling data and input for the model.
- Converting the model into computer readable code that is capable of simulating the system.
- Instructing the computer to perform the simulation correctly and efficiently for various scenarios.
- Summarizing and analyzing the simulation output into performance measures.

Kiviat (1967) was among the first to describe the modeling process and the proficiencies needed. A user with in-depth knowledge of almost any general purpose programming language can provide most of these proficiencies. However, the burden of achieving a final simulation is formidable as the programmer needs to provide software to accomplish all the needed components including random number generation, random variate generation, status updating and time advance, statistical collection and display, etc. To accomplish all this for a single model is generally prohibitive and there are simulation languages and simulation libraries to ease the burden of creating simulations.

While many simulation languages have come and gone, the fundamental simulation approaches are similar. A simulation language executes a model of the system to dynamically act out the behavior of the real system over time. This is done by changing the value of state variables over simulated time. Simulation languages may be generally categorized into two broad families: discrete and continuous. Discrete tools model systems where the state of the system changes in discrete units at specific event times. Continuous tools model systems where the state of the system changes continuously over portions of time. The focus here is on discrete systems, although continuous systems is considered. Nowadays, most simulation languages are multi-modal and support several modeling paradigms which usually mix discrete and continuous capabilities.

2 MODELING WORLD VIEWS

A discrete simulation consists of a set of state variables and a mechanism for changing those variables at event times. A simulation modeling worldview provides the practitioner with a framework for defining a system to accomplish the objectives in sufficient detail that it can simulate the behavior of the system. Unlike simple static descriptive tools such as flow chart (https://en.wikipedia.org/wiki/Flowchart), Visio (https://en.wikipedia.org/wiki/Microsoft_Visio), IDEF (https://en.wikipedia.org/wiki/IDEF), UML (https://en.wikipedia.org/wiki/Unified_Modeling_Language), etc., a simulation modeling worldview must precisely account for the dynamic state transitions that occur over time. The worldview provides a definitive set of rules for advancing time and changing the discrete state (or continuous state) of the model.

Over the 60-year history of simulation we judge four distinct world views dominate in use: event, activities, process, and objects (although other terminology and taxonomies have been offered – (see Overstreet and Nance 1986). These were all developed by the pioneers of simulation during the 1960's (Gordon 1961; Markowitz et al. 1962; Tocher 1963; Dahl and Nygaard 1966) although these world views have been significantly refined over the past 50-year period, the basic ideas have not changed. The evolution of modeling tools has focused on achieving a balance between ease of use and flexibility. The event world view provides the greatest flexibility, but is more difficult to use. Activity based world view gives more context to activities, but with some restriction in modeling. The process view is easier to use, but at the price of decreased modeling flexibility. The object view is the easiest and most natural of all,

but also extracts a price of greater modeling complexity. The history of simulation language development has generally been focused on making the process and object views more flexible, while retaining their advantage with ease of use. At the same time, most modern simulation languages blend their execution methods (using event processing for future events and scan of current events) and employ multi-modal representation (such networks, processes, dynamics), so they do not fall conveniently into a simple taxonomy.

2.1 Modeling with Events

Events are sequential points in time where the system changes state. It becomes the modeler's responsibility to insure that the events are recognized and that the state changes take place. Deciding on the events is a modeler's choice and not always obvious. This choice is governed partly by the behavior of the system, but also by the nature of the performance measures needed from the simulation. Deciding on the events is only part of the challenge. Once an event occurs, it is necessary to map the transition from the present to the successor state. Furthermore there must be some means to serially maintain and add events. And appropriate output must be collected and displayed.

Central to modeling with events is role of the calendar of future events (whose actual data structure has evolved and may vary) that maintains all future events. A discrete event simulation language often works by (1) removing the next event in the event calendar and updating the simulation time to that event time, and (2) then executing state updating procedures associated with that event.

Event logic can be implemented in any general purpose programming language, such as Fortran, C++, C#, etc. However, since a number of the simulation functions such as random variate generation and event scheduling are common to most simulations, using a library of such functions within the context of a programming language greatly facilitates the simulation program development. Examples are the use of Fortran augmented by GASP (Pritsker and Kiviat 1969) and C language programming with CSIM (Schwetman 1986). On the other hand, SIMSCRIPT II.5 (Russell 1983) provided its own programming language to augment its simulation library. It also added basic facilities for defining entities, assigning attributes to entities, and collecting entities into sets.

The event world view tools were widely used during the first 20 years of simulation. These tools were favored by many because they were very flexible and could be used to efficiently model a wide range of complex systems. Note that during this period efficiency was more important because computers had less memory and were significantly slower. However, event-based models were often difficult to understand and debug and required programming proficiency that limited their general use.

Possibly owing to its generality, there are no widely accepted tools for modeling with events. Usually modelers employ flow charts and informal drawings. Some have advocated Petri nets (Haas 2002) and signal flow charts. A general method of modeling discrete events visually are event graphs, which was introduced by Schruben (Schruben 1983). Although the event graph is a valuable tool for understanding discrete event simulation, its use as a general modeling vehicle is quite limited and a number of extensions and embellishments are needed for its general use.

2.2 Modeling with Activities: The Three-Phase Approach

Modeling with a focus on events has dominated the development of simulation models generally within the North American community, while a slightly different approach to discrete modeling called the three-phase approach was initiated by K. D. Tocher in England (see Hollocks 2008). Pidd (Pidd 2004) describes a related method called activity scanning. Activities (operations initiated by events) form the basis of state changes when events take place. The three-phase method has evolved, but essentially consists of (1) advancing time to the next event (called the next (time-dependent) bound event), (2) processing of the one or more next bound events, and (3) processing all other actions that are conditional on the occurrence of the bound events. While a future event calendar is needed, so too are lists of bound

and conditional events. These later condition (state-dependent) event lists are rescanned until no activity may be started.

Central to the three-phase approach is the use of a conceptual modeling tool now called the "Activity Cycle Diagram" (originally called a "wheel chart"). To identify the activities in any system, it is useful to first identify entity types which engage the activities. For example, these might be jobs and an operator. The jobs arrive, wait for the machine, and then leave. The operator sets up the machine and then operates the machine. Otherwise the operator is idle when there are no jobs to be machined. The machining, setup, and arrival are bound activities while the wait and idle are conditional, depending on the bound actions. Here the activity cycle diagram is used to identify the elements of the three-phase simulation.

2.3 Process Modeling

Process modeling, chiefly in the form of GPSS (Gordon 1961), emerged parallel to the development of the event and activity modeling. GPSS, developed at IBM, was a modeling tool as well as a simulation language. It viewed the world as entities (called Transactions) that moved through a model composed of special purpose blocks. Each block had a certain functionality. There is almost a one-to-one correspondence between the block diagram and simulation code. So the simulation model has a visual interpretation as well as a written one. This correspondence is important because the GPSS modeler can essentially translate the visual model almost directly into a "statement" model and instead of language syntax, a visual semantic became the primary means of modeling. This approach meant that the simulation modeler did not need to be a programmer and thus GPSS ushered in a large number of people attempting simulation who would otherwise not participate. GPSS interest was further fueled by the classic book *Simulation Using GPSS* (Schriber 1974) by Tomas J. Schriber at the University of Michigan in 1974. This book was among the first to argue that simulation modeling should be done in a structured fashion.

While GPSS was widely recognized as easy to learn and to use, many questions were raised about its execution efficiency. In GPSS, the simulation executed off two lists: a future events calendar and a current events calendar. As entities are generated they progress through the block diagram as far as they can. If they encounter a block that can schedule its departure some time later, that entity is placed on the future events calendar. However, if the entity cannot proceed (maybe resources are unavailable), then the entity is placed on the current events calendar. The current events calendar is scanned and rescanned until there are no further actions (somewhat like activity scanning) and then the next event is removed from the event calendar and time is advanced. This scanning of the current events calendar, along with a number of other issues, were the basis of a greatly improved version of GPSS in 1983 called GPSS/H (Henriksen and Crain 1983) who were able to shorten execution by a factor of four or more, including the fact that GPSS/H was compiled while the IBM versions were interpreted.

Inspired by the widespread acceptance of GPSS block modeling, a series of new (late 1970s, early 1980s) process languages were developed, based on networks of blocks (nodes). These included Q-GERT (Pritsker 1979) and SLAM (Pegden and Pritsker 1979). Then in the early 1980s the personal computer appeared and SIMAN (Pegden 1982) was among the early simulation languages to execute on this platform. SIMAN also employed stylized blocks to create a model of entities that flowed through a network of individual processing locations.

Process modeling languages generally adopted the use of the future events calendar and the current events calendar. However, in order to reduce the scanning of the current events calendar, efficiencies were added to incorporate a real clock time in place of an integer clock time, and eliminate the scanning of entities and current events whose status will not change (like being in a queue behind others). Other improvements during the 1980s were improved random number and random variate generation, variance reduction, efficient calendar management, and more efficient statistics collection. Few modelers were aware of these since simulation languages became more closely identified with the commercial organizations that developed and maintained them and considered their internal operations to be

intellectual property. The one-to-one relationship of simulation language to commercial organization largely continues today.

One of the important advantages of the process orientation is that the process model is defined graphically in the form of a flowchart (see (Fishman 1973). Hence the user does not have to be a programmer to be able to create a model of the system. Comparing the process model to the event approach, the model logic is much simpler to define and understand/learn, and requires little programming skills. Nevertheless, process simulation languages have inherent limitations since not every situation has a corresponding block in the language. For example, in an emergency room, the patient can be seen as an entity moving through the operation blocks, but there are resources who move about as well who are not well modeled by entities. Eventually, a process modeler "hits the wall" in terms of representation and either must find some less acceptable representation in the current language or find some other way to add the needed capability (usually though programming).

Another conceptual advance that occurred with process modeling was the introduction of hierarchical modeling tools which supported the idea of domain specific process libraries. The basic concept is to allow a user to define their own process steps and blocks by combining existing process steps and blocks. The widely-used Arena modeling system (https://www.arenasimulation.com/) is a good example of this capability. In 1992, Cota and Sargent (Cota and Sargent 1992) added modularization and encapsulation of processes which resulted later in Hierarchical Control Flow Graphs that provided graphical representation (and computation) of simulation models (Sargent 1997). SLX (Henriksen 1995) is an example of more recent process oriented language developed within an object-oriented framework.

2.4 Object-Oriented Modeling

Another view of process modeling is that a model is a set of interacting processes or, more generally, objects. Simula (Dahl and Nygaard 1966), developed in the 1960s, provided an early implementation promoting the idea of objects as elements of the simulation and that these objects can include actions described by logic that control this object and that may interact synchronously or asynchronously with other objects (some ideas evolved from the artificial intelligence language List 2 – (see Nygaard and Dahl 1981). Although developed in the early 1960s as a simulation language , it was rarely recognized in North America and not widely distributed in the US. Part of the reason for its limited exposure was that it was an Algol based simulation programming language so in addition to offering some very unique simulation concepts, it required some unique (for the US) software/hardware. The value of Simula, in hindsight, was a major contribution to simulation, however it was more widely appreciated in the programming community first. With the growth in object-oriented simulation, Simula has been re-discovered as a precursor to many current simulation languages.

Object-oriented simulation modeling generally falls into two camps. The first, exemplified by Simula, is that the simulation language should contain object-oriented concepts that permit the modeler to develop sophisticated simulation programs. In this approach, concepts like abstract data types, inheritance, polymorphism, composition, parameterized types, etc. are relevant because they permit a wide range of objects and behaviors. The models are compact, efficient, and extensible. In other words, they are a better environment for simulation programming. Nowadays, models will typically be built in C++ or Java within the context of simulation packages.

The ideas introduced by Simula provide the foundation for some recent advances made by simulation language designers to make the object-oriented approach to modeling both easy to use and flexible. Although these ideas were introduced as simulation modeling concepts, they have completely changed the design and implementation of programming tools in general. The ideas from Simula directly influenced many later programming languages, including Smalltalk, LISP, C++, Java, and C#. The object-oriented ideas introduced by Simula are not only the most significant development in simulation software, but perhaps the greatest advance in computer science in the last 50 years.

Beyond simulation as a programming challenge, the other object-oriented simulation modeling camp views object-oriented simulation as composed of a wide range of predefined objects, each with a set of behaviors thought to be relevant to the control and use of the objects. This modeling orientation simplifies the model building process by providing a modeling paradigm that is more natural and in many cases easier to use. In an object based approach to modeling we create our model by placing software objects into our model that represent the physical components of the system – e.g. a doctor, a machine operator, a forklift truck, conveyor, etc. These objects may be customized by specifying property values for this object such as service time, travel speed, etc. For example, we model a factory by placing and describing through properties the workers, machines, conveyors, robots, and other objects that make up our factory.

With the process orientation, the modeler describes the actions that take place in the system as entities move through processes. Note that where process steps are described by verbs (seize, delay, release), objects are described by nouns (machine, robot, worker). In the object orientation, the modeler simply describes the physical components in the system, and the behaviors and actions for these objects that are already built into the objects. Hence a worker object has predefined behaviors that allow it to interact with machines and other workers in the model.

It is difficult to envision a more natural way to build a model than by using a collection of prebuilt modeling components that mimic the components in the real system. The challenge with this approach is that if we want to model anything in the real world, we need an extensive library of objects to be able to capture the massive diversity of real objects that you may encounter. For example, it's not adequate to have a single object called robot as there are many different types of robots in the real world. The pursuit by simulation language developers of creating a practical simulation tool based on the object approach illustrates the challenge of having both flexibility and ease of use in the same simulation modeling tool.

Although the flexibility provided by the process orientation makes it still a widely-used approach to simulation modeling, a growing number of successful simulation products have been introduced in the last 20 years based on the object orientation. Just as the second 20 years produced a shift from the event orientation to the process orientation, the past 20 years has seen a shift from the process orientation to the object-orientation. The newer object-oriented tools have a rich set of object libraries that are focused in specific application areas such as manufacturing, supply chain, and healthcare. Some of these tools also allow users to create and customize their own object libraries for specific application areas.

The basic idea of being able to create custom objects as a formal concept was introduced by Ole-Johan Dahl and Kristen Nygaard of the Norwegian Computing Center in Oslo in the 1960's in Simula and Simula 67 (Dahl et al. 1967). Simula introduced the notion of classes of behavior (e.g. Server, Worker, Robot), and instances thereof (objects, e.g. Fred and Drill), as part of an explicit modeling paradigm. A modeler can create an object class such as Car, and then place multiple instances of that class into their model, and customize the behavior of each instance by setting property values. They also introduced the notion of sub-classing objects. This powerful concept allows a user to create a new object class from an existing object class by inheriting, overriding, and extending the object class behavior. For example, a new object class named Truck might be created from the object class named Car by redefining some behaviors, and adding some new behaviors. The ability to create a new object class by starting with an existing object class that has some desired behaviors greatly simplifies the development of object libraries.

Much of the innovative work in simulation language design is occurring in object-oriented simulation tools. These tools are becoming more flexible while retaining their advantage in terms of ease of use, and therefore displacing the process-oriented tools. They also have a key advantage in terms of animation. In the case of the event and process orientations, the addition of animation is a two-step process, where the user first builds the logical model, and then creates the animation as a separate step, and then ties these two components together. In the object-orientation the predefined objects not only come with their associated properties, states, and behaviors, but also their associated 3D animations. This allows the user to rapidly build both the model logic and animation in a single step.

2.5 Systems Dynamic Modeling

Systems Dynamics is a modeling approach developed at MIT during the late 1950's by Jay Forrester (Forrester 1961). It is a form of continuous simulation, where the variables may change continuously with time. Sometimes systems dynamics is used to approximate large scale discrete systems (e.g. modeling populations). In its general form, systems dynamics has a set of state variables that are linked dynamically to a set of differential equations. However, for most applications, the models consist of "levels" and "rates" where the levels are simply state variables and the rates are first order differentials (Sterman 2000). By confining ourselves to this simple form, a wide range of dynamic system problems can be modeled. Systems dynamics are often characterized by Causal Loop Diagrams and by Stock and Flow Diagrams.

A causal loop diagram is a visual means of displaying the system's structure and behavior. However a more detailed model is its stock and flow representation. A stock can be represented as a tank that fills and empties and measures the level of a state variable such as the number of patients in an Emergency Room or the number of people that have been exposed to a disease. A flow is represented as a valve that controls the rate of change of a stock. In this example, the flow of patients to the Emergency Room may be controlled by the extent of insurance. As the number of the insured population increases, the rate of Emergency room visits decreases. But all this may be mitigated by the cost which increases the number of uninsured, which in turn increases the use of the emergency room.

Forrester published the first, and still classic, book in the field titled *Industrial Dynamics* in 1961 (Forrester 1961). One of the best-known Systems Dynamics models is a model of the growth of world population, which was popularized in the bestselling 1972 book *Limits to Growth* (Meadows et al. 1972). This model forecast the exponential growth of population and capital, with finite resources, leading to economic collapse under a wide variety of scenarios. The original model had five levels that measured world population, industrialization, pollution, food production, and resource depletion.

Although any continuous simulation tool can be used to simulate System Dynamics models, a number of specific modeling tools have been developed. For example, (see https://en.wikipedia.org/wiki/Comparison_of_system_dynamics_software). Originally the main simulation language for systems dynamics was Dynamo (now in disuse), but nowadays languages like Stella (https://www.iseesystems.com/store/products/stella-architect.aspx) and Vensim (http://vensim.com/) are popular in the systems dynamics community. Other simulation languages like PowerSim (http://www.powersim.com/), and AnyLogic (http://anylogic.com/) have systems dynamics components, but now offer other combinations with event and process elements. Also although Systems Dynamics models are expressed as continuous systems, most of the applications involve modeling large-scale discrete systems with many entities. An alternative to System Dynamics for large scale discrete systems is agent modeling.

2.6 Agent Based Modeling

Agent Based Modeling (ABM) extends the idea of objects to "agents" whose attributes have a strong association with human behavior, although there is no universal agreement on their definition. This tendency to treat agents as people means that agents need to have intelligent, autonomous characteristics and the capability to make independent decisions. While agents may be independent, they are situated in an environment of other agents and thus there are rules that govern both the individual decision-making as well as interacting with other agents. Generally, the ABM tend to be applied to societal problems which reflect social behavior such as swarming, flocking, following, etc. Some elements of ABM also include systems dynamics.

The basic concept of Agent Based Modeling (see North and Macal 2007) is that a system is modeled by placing agents in the system and letting the system evolve from the interaction of those agents. Each agent is an autonomous entity which interacts with other entities in the system. The focus is on modeling

agent behavior as opposed to system behavior. In a traditional process orientation, entities follow a sequence of process steps, which are defined from the top-down system perspective. In contrast, agent based modeling defines the local behavior rules (often simple) of each entity from a bottom-up perspective. The system behavior is produced as the cumulative result of the agents interacting with each other. Example applications include crowds moving through an area, customers responding to new product introductions, or troops in combat.

The state transition framework for the agents can be modeled using any world view. Agent Based Modeling is often implemented using an object-oriented simulation tool. Hence this is not a new discrete event world view, but rather a group of applications that are often modeled with the object world view. Classes are used to define agent states and behaviors and instances (often large numbers) are placed in the model. The agents (objects) interact and the system state evolves over time. Some of the application areas in Agent Based Modeling present some unique challenges, particularly when large numbers of agents are involved (e.g. modeling the evacuation of fans from a stadium).

For simple applications of Agent Based Modeling the full object-oriented framework with inheritance and sub-classing is sometimes more power than is needed and a simple state diagram may be adequate for defining the class behavior for each agent. The concept of a state diagram was introduced by Shannon and Weaver (Shannon 1949) in their 1949 book *The Mathematical Theory of Communication*. A state diagram defines a finite set of states that the agent can be in, along with the transition conditions that causes a transition between a pair of states. Each diagram typically defines behavior for an object of a given class, and the state transitions for the object instances of that class. Although there are several different variations for state diagrams in use, they all define states as nodes, and arcs for transitioning between states. For example, the state diagram of agents may show how an infected population is processed and how changes in state occur over time.

Because of the increased capacity of computers to manage large numbers of agents, some models that were previously done as discrete approximations using a Systems Dynamics approach are now better done using an Agent Based Modeling approach. This allows greater flexibility in the logic at the expense of a longer execution time. One of the early Agent Based Models was the *Game of Life* developed by John Conway in the 1970's (Conway 1970). The *Game of Life* is a two-dimensional model involving cellular growth that evolves using a fixed time step, where each cell has two states, alive and dead. The state of a cell depends on the state of the neighbors of the previous time step. Conway's game demonstrated the basic concept of the emergence of complexity from simple rules.

Interest in Agent Based Modeling continued to grow and diversify in the 1990s with the appearance of various agent-based tools, particularly Swarm (https://en.wikipedia.org/wiki/Swarm_(app)), NetLogo (https://ccl.northwestern.edu/netlogo/), Recursive Porous Agent Simulation Toolkit (Repast) (http://repast.sourceforge.net/repast_3/), and AnyLogic (http://anylogic.com/).

3 PERSONAL EXPERIENCES IN MODELING: SLAM-SIMAN-INSIGHT-ARENA-SIMIO

Our own experiences with simulation modeling and simulation languages span from the late 1970s to today. While the modeling approaches are different and they extend from the viewpoints discussed earlier, they remain based on the techniques of events and processes with more recent interests in object-oriented methods and multi-paradigm perspectives which we describe in the next section . We present observations for their historical insights.

The development of SLAM originated from a simulation class that was taught by C. Dennis Pegden during the spring of 1978 at the University of Alabama in Huntsville (UAH). Although Pegden had completed a recent class in simulation from A. Alan B. Pritsker during his graduate studies at Purdue, his PhD and focus was on mathematical optimization, so teaching of this graduate simulation class at UAH was both a stretch and learning experience for Pegden. The class had a dozen part-time graduate students, many of whom used simulation daily in their full-time job at NASA and several aerospace contractors. Some of the students were already expert users of GPSS, SIMSCRIPT, and other simulation tools. Given

this impressive background of the graduate students, Pegden decided to leverage the expertise of the students using team teaching and focus on a comparison of the event, process, and object approaches, along with efficient algorithms for implementing simulation tools. During this class Pegden learned about the differences between the event, process, and object approaches and developed the idea of merging together an event and process modeling language into a single framework. Since the GASP IV software was in the public domain, Pegden used GASP IV as the event and continuous components of SLAM, and then extended this framework to include a new process modeling capability. The SLAM language was developed in parallel to and as a direct outcome from teaching this class.

At the time that SLAM was completed, Pritsker and Associates was marketing the GASP IV simulation product, which was developed by Nicholas Hurst as his PhD dissertation under the direction of Alan Pritsker. Since SLAM offered all the functionality of GASP IV, plus the added process modeling capability, it was a clear improvement over GASP IV. In the summer of 1978 Pegden made a trip to Pritsker and Associated and presented the advantages of SLAM, and suggested that Pritsker and Associates market SLAM in place of GASP IV. A marketing agreement was reached, along with an agreement to jointly co-author a new textbook on simulation using SLAM. With the new book and Pritsker's support SLAM quickly became a market success.

Over the next couple of years Pegden shifted his research interest from optimization to simulation and began using SLAM to model complex systems, and quickly realized that although the product worked well for many textbook problems, that the limitations of the process modeling features forced the user to resort back to the event view for most complex real-world applications. Hence the original goal of making a simulation tool that was easier to learn and use for real applications was not being realized for many complex applications – particularly in manufacturing environments. It became clear that a much richer and capable process language would be needed to meet this goal, and that special features were needed to address some manufacturing applications. Hence Pegden decided to start over from scratch and develop the SIMAN language.

Development of the SIMAN language began in the spring of 1981. The focus on the design was both expanding the completeness of the process features as well as improving the execution efficiency. In the summer of 1981 the first IBM PC was released and this became a target platform for SIMAN development, which added additional demands on both memory efficiency and execution speed. In August of 1981 Pegden took a 1-year sabbatical at Tektronix Inc. in Portland Oregon and focused full time on development of the SIMAN language. During this period, several manufacturing processes were successfully modeled using beta versions of the language, which validated the idea that SIMAN provided a significant improvement in process modeling capability. A textbook on simulation using SIMAN was also written during this sabbatical (Pegden 1982).

In the summer of 1982 Pegden contacted Pritsker and Associates to inquire about their potential interest in marketing SIMAN. However, they were heavily invested in SLAM and expressed no interest in the new SIMAN product. Hence in December of 1982 Dennis and Lisa Pegden launched System Modeling Corporation and released the SIMAN language to the market, with instant success. Three key factors that drove its initial market growth were: (1) SIMAN's significantly improved process modeling capability, (2) features focused on manufacturing applications, and (3) SIMAN's ability to run on the new IBM PC. SIMAN incorporated specific constructs to model complex material handling equipment such as AGV's and conveyors. SIMAN also implemented the experimental frame concept promoted by Zeigler (Zeigler 1984) of separating the model logic from the experimentation.

Following the introduction of SIMAN, Systems Modeling Corporation developed the PC-based Cinema animation system (1985) to provide real-time animation of SIMAN models. This provided greater realism than the existing character-based animation systems, and by executing in parallel to the SIMAN model provided a powerful interactive verification/validation platform for models. This was an important step in leveraging the power of animation in simulation. Since Microsoft Windows was not yet available, Cinema was developed using a custom-built windowing system. Since the standard PC had

only 320 by 200 resolution, the initial version of Cinema required a special third-party high-end graphics card and monitor to provide the necessary resolution to produce a quality animation. SIMAN and Cinema were then combined to form the Arena simulation system (1991), and then ported to Microsoft Windows. Arena introduced the concept of graphical processes for model building in place of statements, and provided a dialog box interface for entering properties. It also provided support for both input and output analysis. Arena become widely popular in commercial applications and became the dominant simulation product used by universities for teaching simulation concepts. The popularity within universities was driven by the modeling flexibility of SIMAN, the ease of use of the graphical interface, and the popular "Simulation with Arena" textbooks by W. David Kelton, Randall P. Sadowski, Deborah A. Sadowski, and David T. Sturrock. In 2000 System Modeling Corporation was purchased by Rockwell, who currently markets and supports the Arena product.

Although SIMAN was a step forward in modeling flexibility, there were still many complex applications that were difficult to model with the process view. One issue that occurred frequently was that resources (as well as entities) in SIMAN had no custom decision-making ability. Although entities could seize resources, the resources had no say in the process. In some systems (e.g. healthcare) it's more natural to have resources be in control of their own logic. This challenge of modeling systems requiring complex resource logic became the focus of the work by Stephen Roberts.

Roberts was hired as a faculty member at Purdue University in 1973 and held a joint appointment with the Regenstrief Institute for Health Care, which was situated within the large general and multi-specialty health care clinics for the Indiana University School of Medicine, located in Indianapolis, Indiana. In an analysis of the operating characteristics of those clinics during several years it became clear that only simulation enabled analysis of the complex and stochastic components in those facilities. At Purdue Alan Pritsker and his students were developing network (process) simulation languages based on the discrete simulation portion of GASP IV. While these languages emphasized the flow of entities through processes, it was clear that in health services "patient and related flow" was only part of the environment and patients flow was severely limited by busy health care providers and staff who provided services in a dynamic environment, in contrast to machines and equipment that might be seen in a manufacturing facility. In fact, the flow was almost always constrained by very active human resources who decided what got done and that depended on several factors.

So, during the late 1970s and early 1980 a simulation product called INSIGHT (formerly INS) was developed at the Regenstrief Institute which included the flow of entities (called transaction then), but was oriented around the "job descriptions" as performed by the various "people" resources available. The modeling employed an activity as a service point for entities and resources. The activity specified the number, type, and combination of resources required and the associated activity time, but logical queues in front of an activity held the entities until the service could begin. So, resources served at activities but served entities in queues. Entities did not move into an activity but waited for resources to move them in. Resources were identified individually and optionally by type. Each resource was associated with a primary node in a resource decision tree. Multiple resource decision trees could be defined. That primary node could refer to the queues within the model and/or with other primary nodes. These primary nodes expressed the method of evaluating the associated queues and other nodes by search criteria. For example, a nurse's primary decision node may be to first service the queue of patients in exam rooms needing a nurse service, then service the queue of patients waiting to be placed in an exam room and have vitals take, then execute another primary decision node that says to help in registration or in exit depending on the waiting time of individuals. This decision process could be a part of another decision process or be unique to a resource.

In any case, the resource decision process was dynamic and depended critically on the state of the system. What made the decision process work was the status update algorithm used by the simulation language. Since end of activity events dominated all other events, its updating was central to the modeling and was referred to as a two-stage process. When an activity completed, the resources were put into a

temporary holding state called buffer storage. At this point, Stage 1 was to send the entity on into the network until it progress was halted by the need to acquire resources, start an activity, or leave. When Stage 1 completed, Stage 2 was invoked in which each resource in buffer storage uses its decision tree to determine what it should do next. It either became busy at another activity or became idle. There were specifications that allowed entities to identify and claim resources from buffer storage and to allow the updating of a resource to cause entities to start activities. It should be noted that entities could be delayed because of synchronization requirements such as gathering or remote movement. Further priorities among resources for use at activities could produce pre-emption or resumption of activities.

The INSIGHT simulation language development continued through the 1980s and was described in tutorials at WSC. It never achieved commercial success and the Regenstrief Institute suspended its development when Roberts left for North Carolina State University in 1990. In the 1990s Roberts, along mainly with Jeff Joines, developed a C++ object-oriented simulation language whose process simulation features resembled INSIGHT. Because of the polymorphism of objects, this simulation could employ "teams" as well as individual and type resources in an expanded decision process. Furthermore, the development showed how a language like INSIGHT could be expanded and extended with true object-oriented features. However, use required substantial C++ programming experience, so its acceptance was highly limited and development was phased out in within a decade of its development. Although INSIGHT never achieved commercial success, it highlighted the importance of expanding modeling flexibility beyond simple entity flows to also support the modeling of complex resource logic.

After being away from the simulation business for a couple of years, Pegden was invited to give the Titan talk at the 2005 Winter Simulation Conference on the Future Directions of Simulation. This talk was the impetus to spending time pondering the state of simulation, and what could be done to improve on existing tools. This lead Pegden to develop the concept of modifying the basic object-oriented concepts to replace coded methods with processes, while keeping the basic notion of classes, sub-classes, inheritance, polymorphism, overriding, etc. The motivation for this idea was to bring the benefits of the object-oriented approach using graphical processes, thereby avoiding the need for complex programming skills.

Pegden used these concepts developed for the Titan talk to create the Simio simulation software, which was introduced to the market in 2008. With Simio it became much easier for users to extend and create libraries of objects without the need to code in an object-oriented language such as C++ or Java. Simio also further extended the idea of an intelligent resource as introduced by INSIGHT. In Simio resources (and entities) are also objects that can respond to request using their own custom process logic. This makes it much easier to model situations where the entity or resource is in control of its own behavior.

Another key advance with Simio was the notion of using add-in process for objects to modify their behavior on an instance by instance basis, without modifying the object definition. Before the introduction of this concept into Simio the primary method for adding custom logic to an object instance was by adding custom-coded events to the object. Add-in processes was a significant breakthrough because they are both easier to use as well as more powerful. The added power comes from process logic that can represent activities that span time, whereas a coded event is limited to an instant in time.

The evolution from SLAM – SIMAN – Arena – Simio has occurred over a period of nearly 40 years. During this period the primary modeling paradigm has shifted from events, to processes, to objects. With SLAM the primary modeling approach was events, with processes used where possible. With SIMAN the primary modeling approach was processes with events used when necessary. With Cinema/Arena animation became a central part of the modeling and verification/validation process. With Simio the primary modeling approach is object-based with processes used when needed. This evolution has made simulation easier to use without compromising modeling flexibility.

4 MULTI-PARADIGM MODELING

The earliest simulation modeling approaches were reflections of a single simulation paradigm, namely event, activity, process, and object. In the 1970s there emerged a "combined" paradigm modeling view allowing the user to select the modeling approach that best fits the problem or to combine modeling approaches as the situation dictates. GASP IV (Pritsker 1974) popularized the idea of combining discrete and continuous frameworks in the same model, although there were earlier efforts by others (see, for example Tocher and Splaine 1966). This framework allowed both discrete event modeling and continuous modeling within the same simulation model. It provided both time and state events so that the discrete events could impact the continuous variables and the time events could impact the discrete variables. SLAM (Pegden and Pritsker 1979) advanced the combination to include process modeling and was one of the first widely used tools to promote the idea of mixing alternative modeling approaches (processes and events and continuous) in the same model. In the case of SLAM the process/continuous modeling was used for basic modeling, and the events were used as the "backdoor" to provide added flexibility.

The idea of using simulation in combination with other analytic tools was employed informally early in the history of simulation. In 1983, Shanthikumar and Sargent (1983) offered a unifying definition of hybrid simulation/analytic models. They presented four classes of hybrid models which represent the ways simulation and analytic models could interact. Swinerd and McNaught (2012) recently used a combination of an agent based simulation with an analytic model to create a case of hybrid simulation.

The modern object-oriented simulation tools also employ a multi-paradigm approach. These tools combine the ease and rapid modeling provided by objects with the flexibility added by incorporating user specified events and/or processes. For example, an object representing a server might have selected points in the object logic where the user can insert their own event logic or process logic. Event logic is typically incorporated into objects using either a programming language such as Java or C++, or a special scripting language that can be used in place of code. However, in either case event logic has one major restriction: simulated time cannot advance within the event. This severely restricts the type of logic that can be inserted into an object instance. For example, it is not possible to wait for a worker to become available within an event since this would require time to advance. Simio (http://www.simio.com/index.php) introduced a much more powerful approach which allows users to combine objects with processes. Since processes can span time they provide the user with considerable more power to extend the behavior of their objects. Hence processes can be embedded within an object to wait for a specified time or specified condition (e.g. Fred is available). This approach combines the full power and flexibility of processes with the ease and rapid modeling capability of objects.

AnyLogic (http://anylogic.com/) has combined Agent Based Modeling, Systems Dynamic, and discrete event modeling into a single tool. A single AnyLogic model can combine all three modeling approaches to represent the system behavior.

Also, complementing the interest in multi-paradigm is an interest in unifying simulation modeling within a specific framework or simulation theory. In 1981 Nance (Nance 1981) described the fundamental roles of time and state in discrete event simulations that eventually lead to The Conical Methodology (Overstreet and Nance 1985) which provides a framework for encapsulating the basic modeling viewpoints of event, activity, and process.

The best known simulation theory has been the formalism first proposed by Zeigler in 1984 (Zeigler 1984), now known as DEVS (Discrete Event System Specification). Over the years that theory has been developed and applied to numerous cases (see (Zeigler and Muzy 2017). DEVS is a formal system-theoretic composed of inputs, states, and outputs in combination with time-advance functions to model both discrete and continuous system (potentially hierarchical) components. The abstraction provides a framework that separates modeling and simulation execution to highlight, for example, the ordering and potentially parallel execution of events. The DEVS formalism has attracted world-wide interest and its formulation has hosted a number of extensions and interpretations.

Taking a graph theory approach, Schruben and Yucesan (Schruben and Yücesan 1993) show that an event graph view of discrete event simulation can be used to create a complete and consistent model development (problem-solving) environment. An event graph model is not only visually appealing, but provides a rigorous framework for model building (Savage et al. 2005).

5 MODELING WITH ANIMATION

For the first 30 years of simulation, the output was in the form of textual reports. Although visual animation of simulation execution was envisioned as early as 1970 (Reitman et al. 1970), animation was integrated into commercial simulation in the 1980's. There was significant debate on the merits of animation. Many people felt that animation was a frill that had no real impact on the success of a simulation project. However, animation is now viewed as a critical component of successful simulation projects, and it has a key role in expanding simulation applications within the enterprise. One of the significant benefits of animation is its ability to communicate the model behavior to all the stake holders of the model. Everyone from the shop floor to the top floor can see the behavior that is built into the model. The animation makes the modeling assumptions clear, and is a powerful tool for both verification and validation of the model.

Before animation, decision makers were required to have faith that the model accurately reproduced the behavior of the real system at the appropriate level of detail. There could often be bugs in the model logic, but there was no simple and visual way to detect errors in the logic. With animation, the model comes to life, and the behavior is clearly visible. Animation combined with interactive debuggers have radically improved the ability to find and fix logical errors in a model. Animation not only helps modelers find and fix untended errors in the logic, but also helps to communicate the planned simplifying assumptions that are part of any model. Animation is an essential component for establishing confidence in the model results.

One of the early simulation animation systems was product called See Why developed in the late 80's, which used rudimentary character-based animation. The Witness simulation software (https://www.lanner.com/technology/witness-simulation-software.html) was developed from See Why. The Cinema animation system was a 2D vector-based animation system developed in the same time frame and was a real-time animation system for SIMAN models. The Cinema animation system and SIMAN modeling system were then integrated together into a single platform to create the widely-used Arena product. Another early 2D animation system was Proof (http://www.wolverinesoftware.com/ProofProducts.htm), developed by Jimes O. Henriksen of Wolverine Software in 1989.

In the 90's 3D animation capabilities began to emerge. Some of the early systems included AutoMod (focused on material handling systems) (http://www.appliedmaterials.com/global-services/automation-software/automod), Taylor ED (a precursor to FlexSim (https://www.flexsim.com/)), and Simple++ (a precursor to PlantSim (https://www.plm.automation.siemens.com/)). The animation debate has also extended to the need to have high fidelity 3D animation versus 2D animation. Some argue that 2D animation is adequate to get most of the benefits of animation, and that the extra effort required for 3D brings diminishing returns. Although a higher quality 3D animation does not change the underling model or results compared to a 2D animation, the improved realism can help to communicate the model results. In addition, with the improvements in 3D animation and the wide availability of 3D symbols there is no longer a significant price in time or effort to create the 3D animation. Thus, most modern simulation tools provide a 3D animation environment..

The combination of the object-oriented framework, an immersive 3D modeling environment, and improved 3D drawing features has proven to a very powerful combination for bringing 3D animation into the mainstream of simulation modeling. Most modern simulation tools now provide 3D as a standard capability.

The next wave of animation for simulation is the development of full virtual reality (VR) environments for displaying simulation animations. This allows a viewer to be immersed into the model, and walk through the system in a VR environment that mimics the real system and perhaps interact with the simulation by moving objects, re-directing traffic, or altering simulation activity. Several simulation systems (e.g. Simio, FlexSim, Emulate3D (https://www.demo3d.com/) already support VR environments for viewing animation. Allowing the user to interfere with the animation is the next step in the developing VR display.

6 COMMENTARY

The history of simulation is replete with scores of simulation languages and simulation packages, often each claiming unique modeling components. Many have been lost in the marketplace of popularity due to their inability to capture a simulation audience. Unlike continuous simulation, which is founded in the solution to differential equations, discrete simulations have no common, accepted foundational theory. As a result simulation modeling more often resides in the realm of "art". In practice, the art of simulation requires a rather complete knowledge of what can be simulated by the tool in use, rather than an unbridled focus on the problem being solved. As a result, successful simulations are usually the product of some commercial interest supported by an abundance of documentation and learning tools, all within the framework that promotes the tool benefits.

The translation of a problem concern into a simulation is widely recognized as being an amalgamation of the systems performance objectives, the available input data, and the modeling skills of the modeler. There exists a number of ways to deal with the available input data and diverse performance measures (see (Banks et al. 2010; Law 2015). Model conceptualization has generally relied on graphical representations (block diagrams, flow charts, activity cycle diagrams, and so forth) which are usually associated with specific simulation languages, and there has been limited systematic discussion of general approaches. Robinson (Robinson 2008a, b) presents a two-part conceptual modeling framework. The meaning and requirements of the conceptual model include: validity, credibility, utility, and feasibility. The actual modeling activity is viewed as consisting of: understanding the problem situation, determining the modeling and general project objectives, identifying the model output, identify the model inputs, and determining the model content. Together these provide a framework for conceptualization, but do not as yet yield a computational model. The implementation of the conceptual model must still be implemented by a simulation language or simulation package accessible to the simulation modeler.

The role of the simulation language in simulation modeling remains intertwined. Efforts to make the modeling more transparent and visible through graphical techniques and through natural language offer considerable potential for simulation modeling. Nonetheless the simulation technology must interface with large data storage and analytics to realize its greater potential as computing technology advances.

REFERENCES

Banks, J., J. Carson II, B. Nelson, and D. Nicol. 2010. *Discrete-Event System Simulation*. 5th ed. Upper Saddle River, N.J. ; Singapore: Prentice Hall

Conway, J. 1970. "The Game of Life". *Scientific American* 223 (4):4.

Cota, B. A., and R. G. Sargent. 1992. "A Modification of the Process Interaction World View". *ACM Trans. Model. Comput. Simul.* 2 (2):109-129.

Dahl, O.-J., B. Myhrhaug, and K. Nygaard. 1967. *SIMULA 67 Common Base Language*: Norsk Regnesentral

Dahl, O.-J., and K. Nygaard. 1966. "SIMULA: An ALGOL-Based Simulation Language". *Communications of the ACM* 9 (9):671-678.

Fishman, G. 1973. *Concepts and Methods in Discrete Event Digital Simulation*. New York, United States: John Wiley and Sons Ltd

Forrester, J. 1961. *Industrial Dynamics*. Vol. 1: Cambridge Mass: MIT Press

Gordon, G. 1961. "A General Purpose Systems Simulation Program". In *Proceedings of the December 12-14, 1961 Eastern Joint Computer Conference: computers-Key to Total Systems Control*, edited by W. H. Ware, 87-104. ACM.

Haas, P. J. 2002. "Introduction". In *Stochastic Petri Nets: Modelling, Stability, Simulation*, 1-16. New York, NY: Springer New York.

Henriksen, J. O. 1995. "An Introduction to Slx". In *Proceedings of the 1995 Winter Simulation Conference*, edited by W. R. Lilegdon, D. Goldsman, C. Alexopoulos, K. Kang, 502-509. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Henriksen, J. O., and R. C. Crain. 1983. *GPSS/H User's Manual*: Wolverine Software Corp.

Hollocks, B. W. 2008. "Intelligence, Innovation and Integrity—Kd Tocher and the Dawn of Simulation". *Journal of Simulation* 2 (3):128-137.

Kiviat, P. J. 1967. "Digital Computer Simulation: Modeling Concepts". RM-5378-PR, August, The Rand Corporation.

Law, A. 2015. *Simulation Modeling and Analysis, Fifth Edition*. New York, NY: McGraw-Hill Eduction

Markowitz, H., B. Hausner, and H. Karr. 1962. *SIMSCRIPT: A Simulation Programming Language*. Englewood Cliffs, NJ: Prentice Hall

Meadows, D. H., D. H. Meadows, J. Randers, and W. W. Behrens III. 1972. *The Limits to Growth: A Report to the Club of Rome (1972)*: Universe Books, New York

Nance, R. E. 1981. "The Time and State Relationships in Simulation Modeling". *Commun. ACM* 24 (4):173-179.

North, M. J., and C. M. Macal. 2007. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*: Oxford University Press

Nygaard, K., and O.-J. Dahl. 1981. "The Development of the SIMULA Languages". In *History of Programming Languages I*, edited by L. W. Richard, 439-480. ACM.

Overstreet, C. M., and R. E. Nance. 1985. "A Specification Language to Assist in Analysis of Discrete Event Simulation Models". *Communications of the ACM* 28 (2):190-201.

Overstreet, C. M., and R. E. Nance. 1986. "World View Based Discrete Event Model Simplification". *Modelling and Simulation Methodology in the Artificial Intelligence Era*:165-179.

Pegden, C. D. 1982. *Introduction to SIMAN*: Systems Modeling Corp.

Pegden, C. D., and A. A. B. Pritsker. 1979. "SLAM: Simulation Language for Alternative Modeling". *Simulation* 33 (5):145-157.

Pidd, M. 2004. *Computer Simulation in Management Science*: John Wiley and Sons Ltd

Pritsker, A. A. B. 1974. *GASP IV Simulation Language*: Wiley

Pritsker, A. A. B. 1979. *Modeling and Analysis Using Q-GERT Networks*: Halsted Press

Pritsker, A. A. B., and P. J. Kiviat. 1969. *Simulation with GASP II: A Fortran Based Simulation Language*: Prentice-Hall

Reitman, J., D. Ingerman, J. Katzke, J. Shapiro, K. Simon, and B. Smith. 1970. "A Complete Interactive Simulation Environment GPSS/360-Norden". In *Proceedings of the Fourth Annual Conference on Applications of Simulation*, edited by P. J. Kiviat, M. Araten, 260-270. 807214: Winter Simulation Conference.

Robinson, S. 2008a. "Conceptual Modelling for Simulation Part I: Definition and Requirements". *The Journal of the Operational Research Society* 59 (3):278-290.

Robinson, S. 2008b. "Conceptual Modelling for Simulation Part II: A Framework for Conceptual Modelling". *The Journal of the Operational Research Society* 59 (3):291-304.

Russell, E. C. 1983. *Building Simulation Models with SIMSCRIPT II. 5*. Vol. 5: Caci Los Angeles

Sargent, R. G. 1997. "Modeling Queueing Systems Using Hierarchical Control Flow Graph Models". *Mathematics and Computers in Simulation* 44 (3):233-249.

Savage, E. L., L. W. Schruben, and E. Yücesan. 2005. "On the Generality of Event-Graph Models". *INFORMS Journal on Computing* 17 (1):3-9.

Schriber, T. J. 1974. *Simulation Using GPSS*. New York: Wiley

Schruben, L. 1983. "Simulation Modeling with Event Graphs". *Communications of the ACM* 26 (11):957-963.

Schruben, L., and E. Yücesan. 1993. "Modeling Paradigms for Discrete Event Simulation". *Operations Research Letters* 13 (5):265-275.

Schwetman, H. 1986. "CSIM: A C-Based Process-Oriented Simulation Language". In *Proceedings of the 1986 Winter Simulation Conference*, edited by D. T. Brunner, J. J. Swain, J. M. Charnes, D. J. Morrice, 387-396. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Shannon, C. E. 1949. "W. Weaver the Mathematical Theory of Communication". *Urbana: University of Illinois Press* 29.

Shanthikumar, J. G., and R. G. Sargent. 1983. "A Unifying View of Hybrid Simulation/Analytic Models and Modeling". *Operations research* 31 (6):1030-1052.

Sterman, J. 2000. *Business Dynamics : Systems Thinking and Modeling for a Complex World*. Boston: Irwin/McGraw-Hill

Swinerd, C., and K. R. McNaught. 2012. "Design Classes for Hybrid Simulations Involving Agent-Based and System Dynamics Models". *Simulation Modelling Practice and Theory* 25:118-133.

Tocher, K. 1963. *The Art of Simulation*. Princeton, NJ: Van Nostrand

Tocher, K., and M. Splaine. 1966. "Computer Control of the Bessemer Process". *IRON STEEL INST J* 204 (2):81-86.

Zeigler, B., and A. Muzy. 2017. From Discrete Event Simulation to Discrete Event Specified Systems (Devs). Conference invitee International Federation of Automatic Control,(IFAC), at Toulouse, France.

Zeigler, B. P. 1984. *Multifacetted Modelling and Discrete Event Simulation*: Academic Press Professional, Inc.

AUTHOR BIOGRAPHIES

STEPHEN D. ROBERTS is the A. Doug Allison Distinguished Professor Emeritus in the Edward P. Fitts Department of Industrial and Systems Engineering at North Carolina State University. He has been a faculty member at the University of Florida and Purdue University. He received the Distinguished Service Award from INFORMS College on Simulation, and has served as member, Vice- Chair, and Chair of the Winter Simulation Conference (WSC) Board of Directors representing TIMS (now INFORMS) College on Simulation, and was Secretary, Vice-President/Treasurer, President, and Past-President of the WSC Foundation Board. He was the Proceeding Editor for the 1983 WSC, the Associate Program Chair in 1985, and Program Chair in 1986. His email address is roberts@ncsu.edu.

C. DENNIS PEGDEN is the founder and CEO of Simio LLC. He was the founder and CEO of Systems Modeling Corporation, now part of Rockwell Software, Inc. He has held faculty positions at the University of Alabama in Huntsville and The Pennsylvania State University. He led in the development of the SLAM, SIMAN, Arena, and Simio simulation tools. He is the author/co-author of three textbooks in simulation and has published papers in a number of fields including mathematical programming, queuing, computer arithmetic, scheduling, and simulation. He served as Program Chair for the 1984 Winter Simulation Conference, and was honored with the Winter Simulation Conference Directors' Award (2003). His email is cdpegden@simio.com.