History of Simulation

Evon M. O. Abu-Taieh
The Arab Academy for Banking and Financial Sciences, Jordan

Asim Abdel Rahman El Sheikh
The Arab Academy for Banking and Financial Sciences, Jordan

Jeihan M. O. Abu-Tayeh
Ministry of Planning, Jordan

Hussam Al Abdallat
The Arab Academy for Banking and Financial Sciences, Jordan

INTRODUCTION

The great philosopher Aristotle said, "If you would understand anything, observe its beginning and its development." Therefore, understanding simulation requires observing its history.

Accordingly, simulation can be understood in many ways: "Simulation is the use of a model to represent over time essential characteristics of a system under study" (El Sheikh, 1987). Another definition is "Simulation is the imitation of the operation of a real-world process or system over time" (Banks, 1999).

Simulation was known long before computers. According to Araten et al. (1992), "The first econometrics model of the United States economy was constructed by J. Tinbergen in 1939." Later, as computers developed in the late 1950s and early 1960s, a spawn of computer simulation methodologies and approaches came to life. Computer simulation, like any industry, both affected and was affected by the development of different programming languages and computer capabilities and advances.

This article will first give a background about simulation in general, then it will discuss the classical simulation methodologies. We will address the current trends in simulation by presenting currently used Java-based simulation languages. In this regard, the classical simulation methodologies discussed in this article include the three-phase approach, activity scan, process interaction, event scheduling, transaction flow approach, Petri nets, and Monte Carlo. The languages discussed are simjava, DEVSJAVA, JSIM, JavaSim (J-Sim), JavaGPSS, Silk, WSE (Web-enabled simulation environment), SLX, and SRML (simulation reference markup language). As such, this article will tackle the history of the approaches and methodologies while shedding light on the genealogy of the simulation languages.

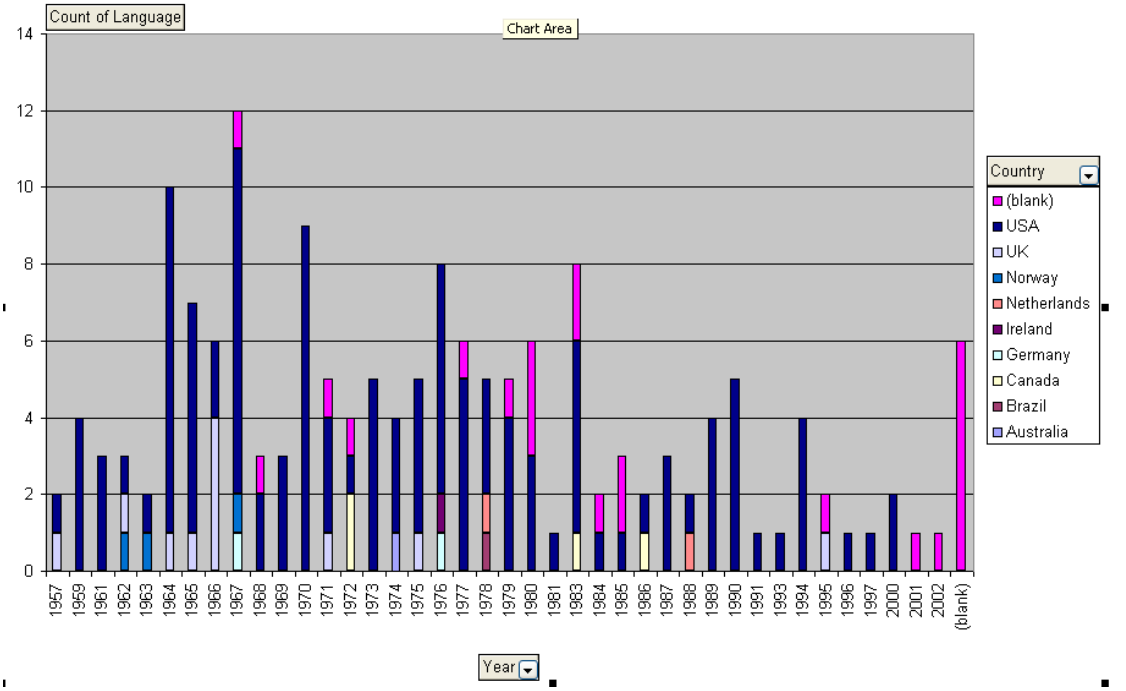
BACKGROUND

Defining simulation in its broadest aspect, we can say that it embodies a certain model to represent the behavior of a system, whether that may be an economic or an engineering one, with which conducting experiments is attainable. When studying models currently used, such a technique enables management and allows us to take appropriate measures and make fitting decisions that would further complement today's growth sustainability efforts, apart from cost decreases as well as service delivery assurance. As such, the computer simulation technique contributed to cost decline, depicted cause and effect, pinpointed task-oriented needs and service delivery assurance, explored possible alternatives, identified problems, proposed streamlined measurable and deliverable solutions, provided the platform for change-strategy introduction, introduced potential prudent investment opportunities, and finally provided a safety net for conducting training courses. Yet, simulation development is hindered due to many reasons. Like a rose, the computer simulation technique does not exist without thorns. Simulation reflects real-life problems; hence, it addresses numerous scenarios with a handful of variables. Not only is it costly and liable to human judgment, but also, the results are complicated and can be misinterpreted.

Within this context, there are four characteristics that distinguish simulation from any other software-intensive work according to Page and Nance (1997). First, simulation uses time as an index variable. Second, one of the objectives in simulation is to achieve correctness. Third, simulation software is involves computational intensiveness. Last but not least, the use of simulation is not typical; in fact, "no typical use for simulation can be described" (Page & Nance, 1997, p. 91).

Hence, there are many methodologies and approaches that simulation practitioners use when working on a simula-

Figure 1. Simulation languages in chronological order



tion project: the three-phase approach, activity scan, process interaction, event scheduling, transaction-flow approach, Petri nets, and Monte Carlo (Abu-Taieh & El Sheikh, 2007). Based on the previously mentioned methodologies and approaches, as well as other programming languages, more than 170 simulation programming languages and more than 60 simulation packages (Abu-Taieh & El Sheikh) were developed, for example, GPSS, GSP, GASP, SIMULA, and so forth. The simulation programming languages are reflected in *Figure **1*** in terms of the date and country of origin for each programming language.

CLASSICAL SIMULATION APPROACHES AND METHODOLOGIES

There are many simulation approaches and methodologies. For this purpose, the most familiar will be thoroughly discussed, namely, the three-phase approach, activity scan, process interaction, event scheduling, transaction flow approach, Petri nets, and Monte Carlo.

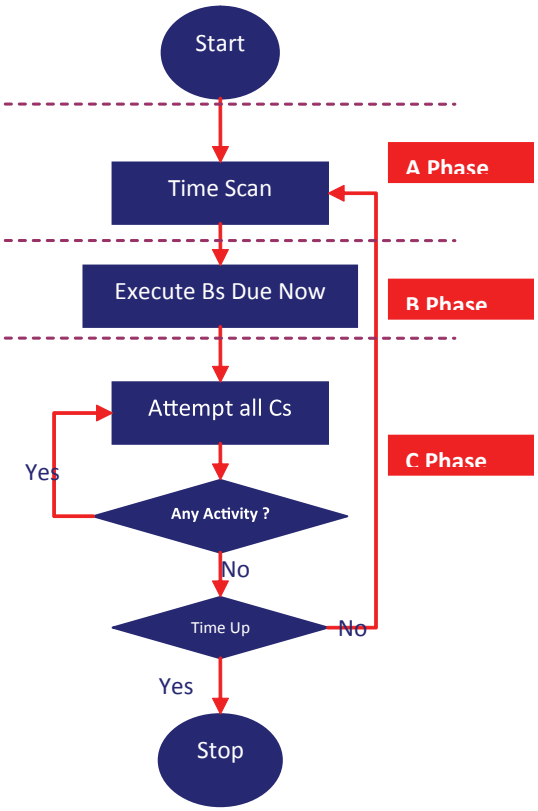
Three-Phase Approach

The first simulation modeling structure is known as the three-phase method. It was described by Keith Douglas Tocker in 1963 in his book *The Art of Simulation* (Odhabi, Paul, & Macredie, 1998), and then discussed in detail by Pidd and Cassel (1998).

Tocker introduced the three-phase approach through the General Simulation Program (GSP), which is considered as the first language effort (Nance, 1995; Pidd, 1998). In 1966, Tocker introduced wheel charts, which were later replaced by the activity cycle diagram and the language CSL (control and simulation language), which represented a simpler approach called activity scan (Nance). Tocker claimed that the structure would enable automatic programming simulation as Nance (1995) states was obvious in the development of OPS-1, OPS-2, OPS-3, and OPS-4, which are simulation languages intended for non-computer-programmers developed by students in MIT.

The three-phase approach has, as the name suggests, three phases—the A phase, B phase, and C phase—as seen in Figure 2. Each phase will be further discussed next.

Figure 2. A three-phase executive (Pidd, 1998)



In the A phase, time is advanced until there is a state change in the system or until something happens next. The system is examined to determine all of the events that will take place at this time (that is, all the activity completions that occur). The A phase is defined formally by Pidd and Cassel (1998): "the executive finds the next event, and advances the clock to the time in which this event is due."

The B phase is the release of those resources scheduled to end their activities at this time. According to Pidd and Cassel (1998), the B phase "executes all B activities (the direct consequence of the scheduled events) which are due at the time."

The C phase starts the activities given a global picture of resource availability. The C phase is defined formally by Pidd and Cassel (1998): "the executive tries to execute all of the C activities (any actions whose start depends on resources and entities whose states may have changed in the B phase)."

In this regard, Michael Pidd (1998) criticized the three-phase model and claimed that this methodology will be less popular as computer power continues to grow, indicating that the reason is "because discrete time slices must be specified. If the time interval is too wide, detail is lost." It is worthwhile to note that Pidd attributed the same criticism to activity scanning, which will be discussed next.

Activity-Scanning Approach

In 1963 the activity-scanning approach was introduced by John Buxton and John Laski (Nance, 1993, 1995). Activity scanning is also known as the two-phase approach. Activity scanning produces a simulation program composed of independent modules waiting to be executed. In the first phase, a fixed amount of time is advanced or scanned. In Phase 2, the system is updated (if an event occurs) as seen in Figure 3. Activity scanning is similar to rule-based programming (if the specified condition is met, then a rule is executed).

Many simulation packages adopted activity scanning, one of which is FirstSTEP.

The activity-scanning approach was represented by CSL, which in turn was influenced by the FORTAN programming language. Later in 1966, Clementson published the research paper *Extended Control and Simulation Language*, which introduced the ECSL simulation programming language. ECSL is based on C.S.L., yet one must note here that CSL is not C.S.L.

In 1978 Parkin and Coats published a research paper titled *EDSIM: Event Based Discrete Simulation using General Purpose Languages such as FORTRAN*. According to Nance (1993, 1995), EDSIM built on ECSL, but reintroduced the FORTRAN-like component.

Figure 3. An activity-scanning executive (Pidd, 1998)

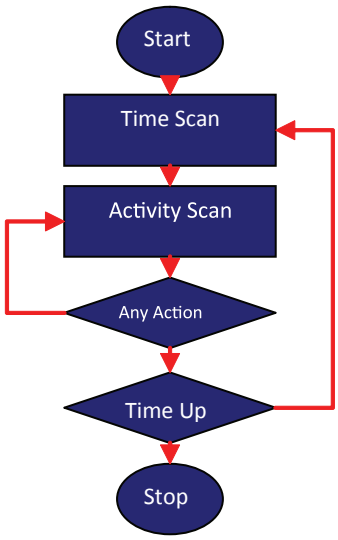
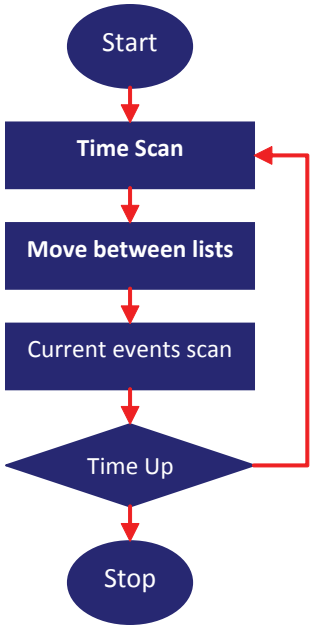


Figure 4. A process-interaction executive (Pidd, 1998)



Process-Interaction Approach

During the early 1960s, the process-interaction approach was introduced in the form of the simulation-oriented language (SOL) and SIMULA. SOL, created by Knuth and McNeley, was an extension of ALGOL, according to Nance (1993, 1995). However, SIMULA was created between 1962 and 1965, and later SIMULA was developed in 1967 by Ole-Johan Dahl and Kristen Nygaard. SIMULA produced CØNSIM (conflict simulator) and DEMOS in the late 1970s (Clema & Kirkham, 1971).

The simulation structure that has the greatest intuitive appeal is the process-interaction method. In this method, the computer program emulates the flow of an object (for example, a load) through the system. The load moves as far as possible in the system until it is delayed and either enters an activity or exits from the system. When the load's movement is halted, the clock advances to the time of the next movement of any load.

This flow, or movement, describes in sequence all of the states that the object can attain in the system as seen in Figure 4. The process-interaction approach was used by many commercial packages, among them, Automod.

Transaction-Flow Approach

The transaction-flow approach was first introduced by GPSS in 1962 as stated by Henriksen (1997). Transaction flow is a simpler version of the process-interaction approach as the following clearly states: "Gordon's transaction flow world-view was a cleverly disguised form of process interaction that put the process interaction approach within the grasp of ordinary users" (Schriber, Ståhl, Banks, Law, Seila, & Born, 2003).

The transaction-flow models consist of entities (units of traffic), resources (elements that service entities), and control elements (elements that determine the states of the entities and resources) as described by Henriksen (1997), Schriber and Brunner (1997), and *GoldSim Web* (n.d.). Discrete simulators that are generally designed for simulating detailed processes such as used in call centers, factory operations, and shipping facilities rely on such an approach, such as in "ProModel, Arena, Extend, and Witness" (GoldSim Web, n.d.). Some scholars (Schriber & Brunner, 1997) it as a "transaction-flow world view" and they add that it "often provides the basis for discrete-event simulation."

In view of the aforementioned, the same scholars continue to describe the best fitted applications to such an approach: "manufacturing, material handling, transportation, health care, civil, natural resource, communication, defense, and information processing systems, and queuing systems in general" (Schriber & Brunner, 1997).

Stock and Flow Approach

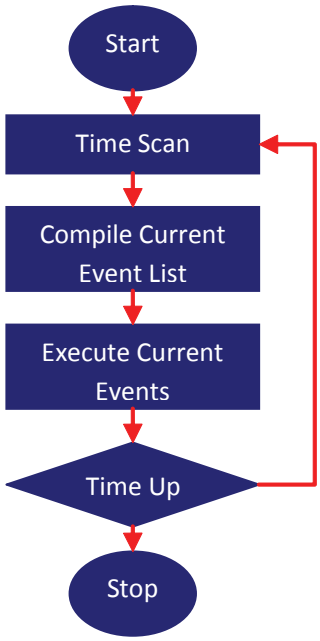
Another approach that is worth mentioning is the stock and flow approach, which is used in dynamic systems. This approach was created at MIT in the '60s by J. W. Forrester (GoldSim Web, n.d.). Stock and flow is based on system dynamics that are built using three principal element types: stocks, flows, and converters.

Event-Scheduling Approach

The event-scheduling approach came into existence through SIMCRIPT in the mid 1960s. SIMSCRIPT spawned from the programming language FORTRAN. The major designer of SIMSCRIPT is Harry Markowitz (Nobel prize winner) and the sole programmer is Bernard Hausner. SIMSCRIPT was fathered by SPS-1 and GEMS, and resembles SEAL (simulation, evaluation, and analysis language). SPS-1 was developed by Jack Little of RAND and Richard W. Conway of Cornell University, and the sole programmer was Bernard Hausner. Many versions came from SIMSCRIPT, like CLP, QUICKSCRIPT, SIMSCRIPTII, SIMSCRIPTII plus, SIMSCRIPTII.5, ESC II, and CSP II, according to Nance (1993, 1995) and Araten et al. (1992).

The basic concept of the event-scheduling method is to advance time to the moment when something happens next (that is, when one event ends, time is advanced to the time of the next scheduled event). An event usually releases a resource. The event then reallocates available objects or

Figure 5. An event-scheduling executive (Pidd, 1998)



entities by scheduling activities, in which they can now participate. Time is advanced to the next scheduled event (usually the end of an activity) and activities are examined to see whether any can now start as a consequence, as seen in Figure 5. The event-scheduling approach has one advantage and one disadvantage as Schriber et al. (2003) states, "The advantage was that it required no specialized language or operating system support. Event-based simulations could be implemented in procedural languages of even modest capabilities." On the other hand, the disadvantage "of the event-based approach was that describing a system as a collection of events obscured any sense of process flow." As such, "in complex systems, the number of events grew to a point that following the behavior of an element flowing through the system became very difficult." Many simulation packages adopted the event-based approach, of which are Supply Chain Builder, Factory Explorer, GoldSim, and ShowFlow.

Petri Nets

Petri nets have been under development since the 1960s, when Carl Adam Petri defined the language in *Kommunikation mit Automaten* (History, 2004): "It was the first time a general theory for discrete parallel systems was formulated."

Furthermore, the idea of Petri nets was developed to answer the question of concurrency, which naturally always arises when discussing simulation. As such, Petri nets handle concurrent discrete events in dynamic systems simulation. As an example, some simulation packages like Optsim (Artifex) use Petri nets.

In order to understand Petri nets, a comprehensive definition must be initially realized. Following are two formal

Table 1. Classical simulation approaches

Approach	Creator	Year	Tool	Language
Three-Phase	Keith Douglas Tocker	1963	wheel charts	General Simulation Program (GSP)
Activity-Scanning Approach	John Buxton and John Laski	1963	activity cycle diagram	CSL
Process-Interaction Approach	Knuth and McNeley Ole-Johan Dahl & Kristen Nygaard	1963		SOL, SIMULA
Event Scheduling	Major Designer: Harry Markowitz (Nobel prize winner) Sole programmer: Bernard Hausner	1963		SIMSCRIPT based on SPS-1 and GEMS
Stock and Flow Approach	Professor Jay W. Forrester at MIT	1960s		GoldSim
Petri Nets	Carl Adam Petri	1960s	tokens	graphical
Monte Carlo Methods (statistical sampling)	Enrico Fermi	1930s	many flavors	

definitions of Petri nets: one that calls it a technique, while the other calls it a language. The first definition follows: "A formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems; a technique undergoing standardization" (PetriNets, 2004).

The second definition states, "Petri Nets is a formal and graphical appealing language, which is appropriate for modeling systems with concurrency….The language is a generalization of automata theory such that the concept of concurrently occurring events can be expressed" (History, 2004).

Petri nets were classified in different ways. The following classifications were listed on *Class Web* (n.d.), among others: Petri net systems of Level 1, Petri net systems of Level 2, and Petri net systems of Level 3. The Level 1 Petri nets systems are characterized by Boolean tokens; on the other hand, Level 2 uses integer tokens. Level 3 is characterized by high-level tokens.

Monte Carlo Methods

At the onset, the Monte Carlo methods were used in the 1930s and denoted generic names (statistical sampling). Enrico Fermi used Monte Carlo to calculate the properties of the neutron. Consequently, during the 1950s, the Monte Carlo methods were used at Los Alamos for the development of the hydrogen bomb. Monte Carlo was popularized and pioneered by Stanislaw Marcin Ulam, Enrico Fermi, John von Neumann, and Nicholas Metropolis (http://en.wikipedia.org/wiki/Monte_Carlo_methods). Another alternative to Monte Carlo is applied information economics (AIE), which is a decision analysis method.

In this context, the Monte Carlo method is formally defined: "Numerical methods that are known as Monte Carlo methods can be loosely described as statistical simulation methods" (*CSEP Web*, 1995). Note that three words stand out in the definition—*loosely, random,* and *statistical*—while statistical simulation is also defined as a "method that utilizes sequences of random numbers to perform the simulation" (*CSEP Web*), as can also be seen within the following definition: "Monte Carlo was coined by Metropolis (inspired by Ulam's interest in poker) during the Manhattan Project of World War II, because of the similarity of statistical simulation to games of chance" (CSEP Web, 1995).

Furthermore, Monte Carlo was referenced in 1987 when Metropolis wrote, "Known to the old guards as statistical sampling: in it new, surroundings and owing to its nature there was no denying its new name of the Monte Carlo Method."

Monte Carlo methods, known for its diversity of applications, are used in nuclear reactor simulation, quantum chromo dynamics, radiation cancer therapy, traffic flow, stellar evolution, econometrics, Dow Jones forecasting, oil well exploration, and VSLI design.

THE JAVA INCLINATION

Since Java came to life in the 1990s, many people from the simulation arena thrived for its utilization. This did not come as a surprise given the history of the object-oriented trail of thought. Additionally, upon examining the process interaction methodology, it was bound to be the basis for today's object-oriented mind-set.

In this context, currently there are several Java-based discrete simulation environments (Kuljis & Paul, 2000): simjava, DEVSJAVA, JSIM, JavaSim (J-Sim), JavaGPSS, Silk, WSE, SLX, and SRML. Following is an elaborated overview.

The first in this list of definitions is simjava, which is based on the process-interaction approach with the purpose of being able to build complex systems. Obviously, simjava is based on the Java programming language, which is inherently objecting oriented (Kuljis & Paul, 2000; Page, Moose, & Gri, 1997).

The second on the list is the DEVSJAVA environment, which was built using Java and is based on the discrete event system specification (Kuljis & Paul, 2000): "A user of DEVSJAVA is able to experiment with any DEVS model from any machine at any time and to interactively and visually control simulation execution."

Likewise, the third on the list is JSIM, which is described by Kuljis and Paul (2000) as a "Java based simulation and animation environment supporting web based simulation as well component-based technology." The component-based technology that JSIM utilizes in this case is Java Beans, as Kuljis and Paul claim in their paper. The idea is to build up the environment from reusable software components that "can be dynamically assembled using visual development tools" (Kuljis & Paul).

In this regard, it is worth noting that JavaSim was later changed to the name J-Sim, which is the fourth definition on the list. J-Sim is considered to be "a set of Java packages for building discrete event process-based simulation" (Kuljis & Paul, 2000). JavaSim was renamed because the word Java is a trade mark owned by SUN Microsystems. J-Sim is an implementation of a simulation tool kit named C^{++}SIM developed in the University of Newcastle (Kuljis & Paul). The official Web site of J-Sim describes it as a "component-based, compositional simulation environment" (*J-Sim Web*, n.d.). Yet, the Web site adds "unlike the other component-based software packages/standards, components in J-Sim are autonomous" (*J-Sim Web*).

The fifth on the list is the JavaGPSS compiler: "The *JavaGPSS* compiler is a simulation tool which was designed for the Internet" (Kuljis & Paul, 2000). JavaGPSS was built so that GPSS can be run on the Internet (Kuljis & Paul).

Within this context, Silk, being the sixth on the list, is defined as the "general-purpose simulation language based around a *process-interaction* approach and implemented

in Java" (Kuljis & Paul, 2000). The purpose of silk is "to encourage better discrete-event simulation through better programming by better programmers" (Kilgore, 2003). According to Healy and Kilgore (1997), "The *Silk* language is an opportunity to make simulation more accessible without sacrificing power and flexibility."

Furthermore, the Web-enabled simulation environment is the seventh item on the list. WSE "combines web technology with the use of Java and CORBA" (Kuljis & Paul, 2000). Also, according to Kuljis and Paul, the WSE environment provides location and distribution transparency, and platform independence.

The eighth on the list, SLX is considered a simulation language that is "C-like," as cited by Henriksen (1997), who first introduced the framework of SLX in 1995 and went on to describe it as a "wolverine software" for the next generation. R. C. Crain discussed SLX in 1997 in the paper entitled "Simulation Using GPSS/H" at the winter simulation conference, and later Henriksen also covered it in his paper entitled "Introduction to SLX" in 1998.

Finally, the simulation reference markup language and the simulation reference simulator were both developed by Boeing and used in many projects (Reichenthal, 2002). Like HTML (hypertext markup language), SRML represents simulation models and as a Web browser represents universal client application. SRML binds the declarative with procedures and like HTML contains both declarative and procedural definitions. According to Reichenthal, "*SRML* is an XML-based language that provides generic simulation markup for adding behavior to arbitrary XML documents."

FUTURE TRENDS

Simulation is not a stand-alone science; it interacts with technology simultaneously as technology advances. There are two major things driving simulation today: the Internet and object-oriented thinking. While the major driving force for simulation is still the need for simulation, the need for virtual reality and augmented reality inter alia demands the prompt advancement of many aspects of simulation. Nevertheless, the elimination of the usual inhibitors like computer hardware speed is yet another prerequisite for advancement in this regard.

CONCLUSION

This article tried to shed light on the numerous facets of simulation history, spanning over the approaches and methodologies of discrete event simulation. The article discussed the most famous discrete event simulation methodologies (three-phase approach, activity scan, process interaction, event scheduling, transaction-flow approach, Petri nets, and Monte Carlo) represented in the programming languages that stemmed from the methodologies. Then the article discussed the future trends through the current history by discussing nine Java-based languages.

REFERENCES

Abu-Taieh, E., & El Sheikh, A. (2007). Commercial simulation packages: A comparative study. *International Journal of Simulation, 8*(2), 1473-804x.

Araten, M., Hixson, H. G., Hoggatt, A. C., Kiviat, P. J., Morris, M. F., Ockene, A., et al. (1992). The Winter Simulation Conference: Perspective of the founding fathers. In J. J. Swain, D. Goldsman, R. C. Crain, & J. R. Wilson (Eds.), *Proceedings of the 1992 Winter Simulation Conference.*

Banks, J. (1999, December 5-8). Introduction to simulation. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, & G. W. Evans (Eds.), *Proceedings of the 1999 Winter Simulation Conference*, Phoenix, AZ (pp. 7-13). New York: ACM Press.

Class Web. (n.d.). Retrieved April 1, 2004, from http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C10/10-7.html

Clema, J., & Kirkham, J. (1971). CONSIM (conflict simulator): Risk, cost and benefit in political simulations. In *Proceedings of the 1971 26th Annual Conference* (pp. 226-235). New York: ACM Press.

El Sheikh, A. (1987). *Simulation modeling using a relational database package.* Unpublished doctoral dissertation, The London School of Economics, London.

GoldSim Web. (n.d.). Retrieved September 1, 2003, from http://www.goldsim.com

Healy, K. J., & Kilgore, R. A. (1997, December 7-10). Silk: A Java-based process simulation language. In S. Andradóttir, K. J. Healy, D. Withers, & B. Nelson (Eds.), *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA (pp. 475-482). New York: ACM Press.

Henriksen, J. (1997, December 7-10). An introduction to SLX™. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA (pp. 559-566).

History. (2004). Retrieved April 18, 2004, from http://www.daimi.au.dk/PetriNets/faq/

J-Sim Web. (n.d.). Retrieved April 10, 2004, from http://www.j-sim.org

Kilgore, R. (2003, December 7-10). Object-oriented simulation with SML and silk in .Net and Java. In S. Chick, P. J. Sánchez, D. Ferrin, & D. J. Morrice (Eds.), *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, LA (pp. 218-224).

Kuljis, J., & Paul, R. (2000, December 10-13). A review of Web based simulation: Whither we wander? In J. A. Joines, R. R. Barton, K. Kang, & P. A. Fishwick (Eds.), *Proceedings of the 2000 Winter Simulation Conference*, Orlando, FL (pp. 1872-1881). San Diego, CA: Society for Computer Simulation International.

Metropolis, N. (1987). The beginning of Monte Carlo method. *Los Alamos Science, 15*, 125-130. Retrieved April 18, 2004, from http://jackman.stanford.edu/mcmc/metropolis1.pdf

Nance, R. (1993). A history of discrete event simulation programming languages. *ACM SIGPLAN Notices, 28*(3), 149-175.

Nance, R. (1995). Simulation programming languages: An abridged history. In C. Alexopoulos, K. Kang, W. R. Lilegdon, & D. Goldsman (Eds.), *Proceedings of the 1995 Winter Simulation Conference.*

Nance R. (1996). A history of discrete event simulation programming languages. In *History of programming languages* (Vol. 2). New York: ACM Press.

Odhabi, H., Paul, R., & Macredie, R. (1998, December 13-16). Making simulation more accessible in manufacturing systems through a "four phase" approach. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M. S. Manivannan (Eds.), *Proceedings of the 1998 Winter Simulation Conference*, Washington, DC (pp. 1069-1075). Los Alamitos, CA: IEEE Computer Society Press.

Page, E., Moose, R., & Gri, S. (1997, December 7-10). Web-based simulation in Simjava using remote method invocation. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA (pp. 468-474). New York: ACM Press.

Page, H., & Nance, R. (1997). Parallel discrete event simulation: A modeling methodological perspective. *ACM Transactions on Modeling and Computer Simulation, 7*(3), 88-93.

PetriNets. (2004). Retrieved April 18, 2004, from http://www.petrinets.info/graphical.php

Pidd, M. (1998). *Computer simulation in management science* (4th ed.). Chichester, England: John Wiley & Sons.

Pidd, M., & Cassel, R. (1998, December 13-16). Three phase simulation in Java. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M. S. Manivannan (Eds.), *Proceedings of the 1998 Winter Simulation Conference*, Washington, DC (pp. 267-371). Los Alamitos: IEEE Computer Society Press.

Reichenthal, S. (2002, December 8-11). Re-introducing Web-based simulation. In E. Yücesan, C.-H. Chen, J. L. Snowdon, & J. M. Charnes (Eds.), *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA (pp. 847-852).

Schriber, T., & Brunner, D. (1997, December 7-10). Inside discrete-event simulation software: How it works and why it matters. In S. Andradóttir, K. J. Healy, D. H. Withers, & B. L. Nelson (Eds.), *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA (pp. 14-22).

Schriber, T. J., Ståhl, I., Banks, J., Law, A. M., Seila, A. F., & Born, R. G. (2003, December 7-10). Simulation text-books: Old and new (panel). In S. Chick, P. J. Sánchez, D. Ferrin, & D. J. Morrice (Eds.), *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, LA (pp. 238-245).

KEY TERMS

Continuous Simulation Systems: These are systems that deal with time as a continuous function, for example, when simulating the flow of water from a reservoir.

Discrete Simulation Systems: These are simulation systems that treat the time variable as a discrete variable. Usually any systems that deal with queues (supermarkets, banks) are of discrete nature.

Monte Carlo Methods: These are statistical simulation methods.

Petri Nets: Petri nets are "a formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems; a technique undergoing standardization" (PetriNets, 2004).

Simulation: It "is the imitation of the operation of a real-world process or system over time" (Banks, 1999).

Simulation Methodology: It is a world view of abstracting the real world and mapping it into a computer program.