

GENERAL-PURPOSE SYSTEMS FOR EFFECTIVE CONSTRUCTION SIMULATION

By Julio C. Martinez¹ and Photios G. Ioannou², Members, ASCE

ABSTRACT: This paper examines the characteristics of discrete-event simulation systems in terms of their application breadth (general or special purpose), modeling paradigm (process interaction versus activity scanning), and flexibility (programmable or not). Several construction simulation systems are examined with primary emphasis on CYCLONE and STROBOSCOPE as representatives of the wide range of tools that are currently available. CYCLONE is a well-established, widely used, and simple system that is easy to learn and effective for modeling many simple construction operations. STROBOSCOPE is a programmable and extensible simulation system designed for modeling complex construction operations in detail and for the development of special-purpose simulation tools. The characteristics of these systems, as well as other recent developments, illustrate that an effective general-purpose simulation tool for construction is in essence one based on extended forms of activity cycle diagrams and the activity scanning modeling paradigm. As explained through several examples, these representations are indeed the most convenient and intuitive for construction simulation systems. Furthermore, the programmability of such a system is the principal factor that determines its power, flexibility, and ease of learning and use.

INTRODUCTION

Discrete-event simulation has been used to analyze and design construction operations for over three decades. This extensive research activity has created an implicit but limited notion of what constitutes a "construction simulation system." It has also created an implicit understanding that, conceptually, these systems are indeed better suited to construction operations modeling than systems commonly used in other industries. A clear and explicit explanation of the essence of a construction simulation system and the reasons for which these systems are a natural choice for construction modeling, however, are not available. This paper explains in detail the essence of a construction simulation system as well as the reasons that make such systems intuitively ideal for construction operations modeling.

This sets the background for a review of the various systems that have been designed for modeling construction operations. Although a representative set of tools is covered, the emphasis of this paper is on CYCLONE (Halpin and Riggs 1992) and STROBOSCOPE (Martinez 1996). CYCLONE is a well-established, widely used, and simple system that is easy to learn and effective for modeling relatively simple construction operations. STROBOSCOPE is a programmable and extensible simulation system designed for modeling complex construction systems in detail and for the development of special-purpose simulation tools.

SIMULATION TOOL CHARACTERISTICS

The scope, flexibility, and suitability-to-task of a simulation tool depends on its application breadth, modeling paradigm (simulation strategy), and flexibility.

Application breadth is the scope of models for which the tool is designed. General-purpose simulation tools and languages target a very broad domain and can be used to model almost any type of operation. General-purpose simulation tools

for construction modeling include those described in Halpin (1976), Chang (1986), Paulson et al. (1987), Ioannou (1989), Mohieldin (1989), Liu (1991), Odeh (1992), and Sagert (1995). In contrast, special-purpose simulators are tools that target a narrow domain such as ductile iron pipe installation. Special-purpose simulators designed for specific construction tasks include those described in McCahill and Bernold (1993), Shi and AbouRizk (1997), Oloufa and Ikeda (1997), and Martinez (1997, 1998).

Modeling paradigm or simulation strategy is the conceptual framework that guides model development and determines how the modeler views the system being modeled (Hooper 1986; Balci 1988). The two main simulation strategies are process interaction (PI) and activity scanning (AS). Event scheduling (ES) is a third simulation strategy that is often combined with PI or AS. (Some authors also consider a transaction-based approach that is very similar but subtly different to PI. In the following, all statements about PI also apply to the transaction-based approach.)

Flexibility reflects the capability of the tool to model complex situations and to adapt to a wide range of application requirements. Advanced simulation systems typically involve computer programming and are flexible enough to model very complex operations in detail. Simpler nonprogrammable tools are typically easier to learn and can be used to model many simple operations effectively. However, they often require assumptions that prevent the effective analysis of many complex or detailed operations.

Simulation Strategy

By far the most significant characteristic of any general-purpose discrete-event simulation system is its simulation strategy. The main simulation system strategies in use today for modeling construction processes are PI and AS.

A PI model is written from the point of view of the entities (transactions) that flow through the system. These entities typically arrive, undergo some processing where they capture and release scarce resources, and then exit. This strategy is particularly suited to modeling operations where the moving entities are differentiated by many attributes and where the machines or resources that serve these entities have few attributes, a limited number of states, and do not interact too much (Hooper 1986). Most operations in manufacturing and the industrial and service industries are of this type. Consequently, a large number of commercial simulation tools are based on the PI para-

¹Asst. Prof. of Civ. Engrg., Virginia Polytechnic Inst. and State Univ., 200 Patton Hall, Blacksburg, VA 24061-0105.

²Assoc. Prof. of Civ. and Envir. Engrg., Univ. of Michigan, Ann Arbor, MI 48109.

Note. Discussion open until January 1, 2000. To extend the closing date one month, a written request must be filed with the ASCE Manager of Journals. The manuscript for this paper was submitted for review and possible publication on July 7, 1998. This paper is part of the *Journal of Construction Engineering and Management*, Vol. 125, No. 4, July/August, 1999. ©ASCE, ISSN 0733-9634/99/0004-0265-0276/\$8.00 + \$.50 per page. Paper No. 18732.

digm (e.g., GPSS, SIMAN, SLAM, ProModel, SimScript, ModSim, Extend, etc.)

In contrast, AS models are written from the point of view of the various activities that are performed and focus on identifying these activities and the conditions under which they take place. Three-phase AS is a modified approach that incorporates ES concepts to increase performance (Tocher 1963). AS is particularly adept at modeling systems with interdependent components subject to complex activity start-up conditions where many resources with distinct properties must collaborate according to highly dynamic rules (Hooper 1986). Most AS languages were developed in the 1960s in Europe. They include GSP (Tocher and Owen 1960), CSL (Buxton and Laski 1962), and HOCUS (Hills 1971).

Simulation strategies are independent of other tool characteristics such as object-orientation or hierarchical decomposition. Simulation strategies also give special-purpose simulators their unique flavor and shape simulation model development, even when using general-purpose programming languages (Balci 1988). The strategy used by a simulation tool, whether it is PI or AS, has a strong impact on the thought process that leads to model development as well as on the way a model is presented to the computer (Evans 1989). For this reason, the superiority of one strategy over another has been the subject of much discussion and has led to several comparisons over the years (Hills 1973; Zeigler 1976, Hooper and Reilly 1982; Birtwistle et al. 1985; Hooper 1986). Overall, these investigations have concluded that all strategies are equally general and powerful in terms of being or not being able to represent specific problems. Particular strategies, however, make modeling certain classes of problems easier and this makes them more suitable for certain tasks. In fact, one of the main objectives of this paper is to illustrate that in construction AS is indeed a more natural and effective strategy than PI.

TABLE 1. Activities, Required Conditions, and Outcomes for Scraper and Pusher Operation

Conditions needed to start (1)	Activity (2)	Outcome of activity (3)
Pusher at push-point	<i>PushLoad</i>	Pusher ready to backtrack Loaded scraper ready to haul
Scraper at cut		
Pusher ready to backtrack	<i>Backtrack</i>	Pusher at push-point
Loaded scraper ready to haul	<i>Haul</i>	Loaded scraper ready to dump
Loaded scraper ready to dump	<i>DumpAndSpread</i>	Dumped soil Scraper ready to return to cut
Scraper ready to return to cut	<i>Return</i>	Scraper at cut

Effect of Simulation Strategy on Model Development

The effect of simulation strategy on model development in construction is best illustrated by using both PI and AS to model the same process. The example chosen for this comparison is a simple earth-moving operation where a pusher and scraper work together to push-load scraped material into the scraper's bowl. The pusher then backtracks to get ready to push-load again, and the scraper hauls, dumps, spreads, and then returns for another push-loading. A more complex version of this problem will also be used to illustrate why AS is the natural approach for modeling complex construction operations in detail.

AS Model for Simple Scraper and Pusher Operation

The first step in developing an AS simulation model is to identify the various activities that can take place, the conditions necessary for these activities to start, and their outcomes when they end. Start-up conditions and outcomes are typically best described in terms of the resources involved and their state (i.e., their condition). Table 1 summarizes the activities for this operation, the conditions under which they can start, and their outcomes. Activity *PushLoad*, for example, can start when a pusher is waiting at the pushing point and a scraper is waiting at the cut. Its outcomes are a loaded scraper ready to haul and a pusher ready to backtrack.

The conditions necessary for activities to start are often the outcomes of other activities. This makes it easy and natural to represent the information in Table 1 using an activity-oriented network as shown in Fig. 1. This type of network is called an activity cycle diagram (ACD) and consists of alternating circles and rectangles connected with links. The rectangles are called activities and represent tasks performed by one or more resources. The circles are called queues and represent inactive resources in specific states. Each queue in Fig. 1 corresponds to an entry in Table 1 under the columns "Conditions needed to start" or "Outcome of activity." Similarly, each activity in Fig. 1 corresponds to a row in Table 1. Queues and activities are connected with directional arcs that indicate whether the resources in a queue are required to start an activity (from the queue to the activity) or are the result of an activity (from the activity to a queue). In practice, an ACD can be constructed without the need to first create a table. A common approach is to draw the activity cycles of the different resources separately and then combine the cycles into a network by joining them where they have activities in common (Halpin and Riggs 1992; Szymankiewics et al. 1987).

ACDs are intuitive representations and their activity-oriented paradigm makes them easy to communicate to persons familiar with other modeling tools [such as critical path method precedence networks]. ACDs evolved from wheel charts that were used as blueprints for the construction of GSP simulation programs, much as flowcharts are used to construct

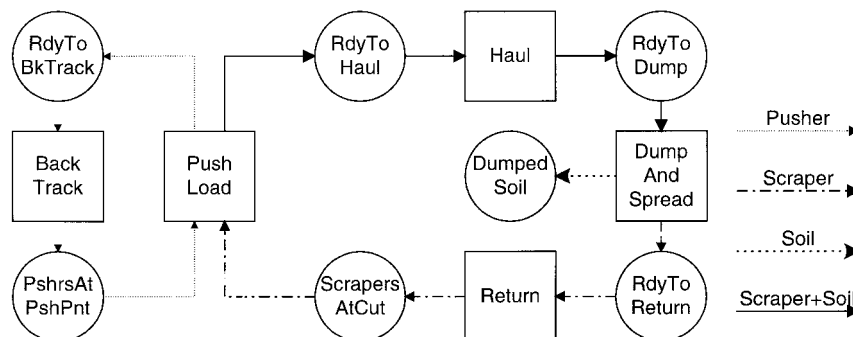


FIG. 1. ACD for Scraper and Pusher Operation

conventional programs (Tocher 1964). ACDs were made popular by the HOCUS simulation system in the late 1960s (Hills 1971). The familiar CYCLONE networks used extensively for construction simulation modeling over the last 20 years represent yet a further extension and refinement of ACDs as do the networks used by RESQUE (Change 1986), COOPS (Liu 1991), CIPROS (Odeh 1992), and STROBOSCOPE (Martinez 1996).

ACDs are a natural means for representing three-phase AS simulation models at the conceptual level and serve as a high-level representation of the main elements of the simulated process. Typically, it is neither necessary nor convenient for an ACD to show more detail as this may obscure the main model structure. An AS model uses the ACD as a blueprint for developing a detailed computer program that repeatedly checks for the conditions necessary for each activity to start. When the conditions are satisfied, the program performs the appropriate actions. These actions typically include acquiring the resources that enabled the activity to start, determining how long the activity will last, holding the acquired resources for the duration of the activity, and releasing the resources when the activity ends.

In some cases, all of the conditions needed for an activity to start are provided by the completion of another activity. The only condition for Activity *Haul* to start, for example, is the availability of a loaded scraper ready to haul. This condition also happens to be one of the outcomes of Activity *PushLoad*. Consequently, the conditions needed for Activity *Haul* to start are always and immediately satisfied every time Activity *PushLoad* ends. In three-phase AS terminology, the start of Activity *Haul* is bound to the end of Activity *PushLoad*. Three-phase AS takes advantage of this observation to speed up program execution by distinguishing between conditional and bound activities. The three-phase approach is more efficient than pure AS because the simulation program does not have to scan bound activities for start-up conditions—they simply start whenever the activity to which they are bound ends. In the ACD of Fig. 1 all activities are bound except for *PushLoad*, which is conditional.

In addition to representing activities, an AS simulation model must also model the resources involved in the process, such as materials, labor, and equipment. Resources and their state are equally important because they constitute the predominant requirement for activities to take place and because the primary effect of activities is to change their state. In AS all resources are considered equal. No distinction is made between those that serve and those being served or between materials, labor, and equipment. In the pusher and scraper operation, for example, except for the fact that each represents a different resource, pushers, scrapers, and soil are all treated alike. As described below, this equitable modeling approach is in direct contrast to that taken by PI and is one of the main

reasons why AS is more natural for modeling construction operations.

PI Model for Simple Scraper and Pusher Operation

The first step in developing a PI simulation model is to identify the entities that flow through the system and the scarce resources for which these entities compete. This is particularly important because, typically, moving entities cannot compete for the acquisition of other moving entities, and scarce resources (being completely passive) cannot compete for other resources. Moreover, a PI model must adopt the point of view of the moving entities and this makes certain choices much more convenient than others. Thus, more than anything else, the prudent choice of moving entities and scarce resources determines the ease of model development as well as the effectiveness of the resulting simulation model. This choice, however, is by no means obvious and as shown below it often makes modeling the dynamic interactions that occur in construction quite challenging.

In the scraper and pusher operation, for example, it is possible to think of the scrapers, the pushers, or the soil as moving entities. It is also possible to think of the scrapers or the pushers as scarce resources. Fig. 2 illustrates a PI model where the soil is the main moving entity and the scrapers and pushers are scarce resources. Notice that because PI must be triggered by moving entities, it is necessary to clone (split) each unit of soil (a moving entity) to model the return of a scraper and pusher back to the loading area correctly.

At first glance, the PI model shown in Fig. 2 may appear to be appropriate. This would be the case if units of soil arrive to the system at some rate, wait for processing, and then leave (which is what typically happens in a manufacturing scenario). In the scraper and pusher operation, however, all soil units exist simultaneously from the very beginning of the process. Thus, a computer program implementing this PI model would have to spend a significant amount of time constantly checking to see if each separate unit of soil (the moving entity) can advance to the next step in its life. Furthermore, modeling and storing each scraper-full of soil as a separate entity would waste a significant amount of computer memory without providing any benefits.

A different PI model is shown by the flowchart in Fig. 3, where scrapers are the main moving entities and pushers are the scarce resources (in this case soil is modeled as a secondary moving entity). This flowchart is in GPSS notation (Schriber 1990) because there is no standard way of representing a PI model. Flowcharts using other notations (e.g., SIMAN) or PI-oriented networks (e.g., SLAM) use slightly different shapes and names for the blocks, but are conceptually similar. For this earth-moving example, the processes followed by scrapers and pushers are cyclic, and therefore it is necessary to create clones regardless of which resources are chosen as

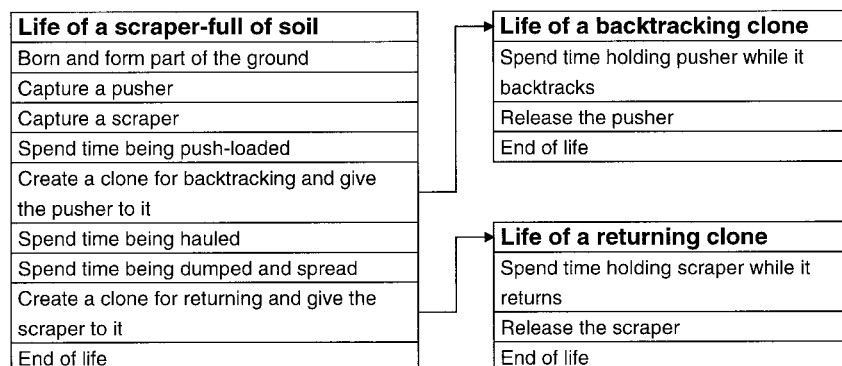


FIG. 2. Moving Entities in PI Model of Scraper and Pusher Operation

ditions at the cut. Activity *ToSide* may be of higher priority when the amount of side in need of trimming is excessive. Activity *BackTrack* may have higher priority when several scrapers are idle waiting to be push-loaded. (For simplicity we are assuming that the excavation material is hard and does not allow scrapers to self-load or push-pull, thus making push-loading necessary.) When there is side in need of trimming, Activities *ToPushArea* and *TrimSide* compete for a pusher at the side. In both cases, the details for determining the conditions under which one activity has priority over another are not shown in the ACD.

For the revised version of the earth-moving operation shown in Fig. 4, idle pushers can be in three different states: (1) At the pushing point; (2) ready to backtrack; (3) or at the side. In the previous version shown in Fig. 1 the ready-to-backtrack state could be disregarded because Activity *Backtrack* was bound to Activity *PushLoad*. This is no longer the case and the number of meaningful nonactive states for pushers has increased from one to three.

Attempting to represent the new problem with a PI model is significantly more difficult. A scraper must now capture a pusher when the latter is idle at the pushing point. The pusher, however, could be idle at the side or idle and ready to backtrack and thus not be suitable for capturing by the scraper. It suddenly becomes more convenient to switch the point of view of the entire PI model and treat pushers as moving entities and scrapers as scarce resources.

Modeling pushers as moving entities makes it necessary to resolve how to route a moving pusher after it push-loads, or when it is at the side, while at the same time managing scrapers as scarce resources. A pusher must capture a scraper before push-loading but must not relinquish it when done. Instead, the pusher must split and create a clone that will haul, dump, return, and then relinquish the scraper. Furthermore, several other resource interaction issues must be resolved before a valid PI model can be created.

Although this example is small, it does illustrate the difficulties associated with using PI to model even simple construction processes (the complete PI model for this example is not shown here due to space limitations). In particular, switching the roles of pushers and scrapers as moving entities and scarce resources requires a shift in modeling paradigm that is far from obvious. As a construction operation becomes more detailed and the number of interacting resources increases, the convenience of simulation modeling using PI decreases and the advantages of AS become more significant.

It is common in construction for several resources to collaborate to accomplish a task. In most cases, an activity cannot start unless all of these resources are available and in the correct state. If any of the resources is not available or in the proper condition then the task cannot be carried out. Situations such as these are naturally and automatically represented by ACDs, but pose difficulties for PI tools. As an example, consider the PI model of Fig. 2, where the main moving entity is a scraper-full of soil. The soil first tries to capture a pusher and then a scraper. If a pusher is available but a scraper is not, the soil captures a pusher and then waits until a scraper becomes available. During this waiting time the pusher is committed and cannot do anything else. That is, it cannot be captured by a moving entity in another process where a pusher does not need to collaborate with a scraper (such as when "a side in need of trimming" tries to capture it to trim the sides). Halpin describes this type of problem in detail (1973).

Although some of the newer PI tools (e.g., ProModel) have features that enable all-or-none resource acquisition, they still force the differentiation between moving entities and scarce resources. When many resources interact, one must be treated as a moving entity and all of the others as scarce resources.

Dealing with the multiple states of all these resources decreases the convenience of using a PI tool geometrically.

Having many interacting resources in a PI model can also lead to the problems of entanglement and deadlock (Tocher 1979). Entanglement is a complex problem that occurs when the processes followed by several moving entities of different types get intertwined. Deadlock occurs when a moving entity captures one scarce resource (e.g., a scraper) and is waiting to capture a second resource (e.g., a pusher), while another entity has captured the resource needed by the first entity (i.e., the pusher) and is waiting to capture the resource in possession of the first entity (i.e., the scraper). Neither entity can continue its path because each has the resource that the other needs.

This brief comparison of simulation strategies indicates that AS makes it much easier to model complex processes with many interacting resources. The primary reasons for this can be summarized as follows

- The fundamental point of view of a model does not shift as the number of resources or the complexity of the problem increases.
- There is no need to differentiate between scarce resources and moving entities.
- The all-or-nothing requirement for resource acquisition (which is common in construction) is satisfied automatically.
- It is not necessary to be concerned about entanglement or deadlock because they cannot occur. (When modeling complex construction processes using PI tools these must be explicitly avoided.)
- In construction the focus is on the tasks that need to be performed and their requirements and not on what each resource needs to do next.

Simulation System Flexibility and Programmability

In a general sense, the flexibility of a general-purpose simulation system is independent of the strategy it adopts, whether it is PI or AS. Programmability, however, significantly increases flexibility.

It is possible to provide simulation systems with a certain level of flexibility by building capabilities into the tool. Once a capability is built-in, it can be used within the limitations of its design. Consider, for example, a discrete-event simulation tool with the capability to model learning-curve effects by adjusting the parameters of an activity's duration as a function of the number of times the activity has been performed. For instance, the parameter for the duration of the n th activity occurrence P_n could be a function of the parameter for the duration of the first occurrence P_1 and a constant s , $P_n = P_1 \cdot n^s$. Such a tool may have a facility or a template where the user can indicate that learning effects should be considered and where the values of P_1 and s are specified. The addition of this template gives the tool the flexibility to model learning-curve effects, but at the same time limits its capabilities to this particular learning model. Furthermore, the addition of this template increases the complexity of the tool because more has to be learned before it can be used.

Instead of continuously adding template-based capabilities to a simulation system, it is possible to attain much more flexibility by providing the system with end-user programmability. (This obvious strategy has been currently adopted by many software products, particularly those that support a version of the BASIC language.) As an example, consider the case of a simulation tool that supports the simplest level of programmability, the ability to use a formula to define the parameter for the duration of an activity. It is then possible to simply write the appropriate formula as part of the model and achieve the same learning-curve effect described above. For example,

consider a steel erection activity named *Erect* for which $P_1 = 11.8$ h and $s = -0.129$. The parameter for the duration of the activity could then be specified as $11.8 * Erect.TotInst - 0.129$, where *Erect.TotInst* is a variable that dynamically returns the number of times that the *Erect* activity has been performed. Thus, simple programming can be used to represent not just this but any other learning-curve model as well. Moreover, it can be used to model any number of other situations that require the use of nonstationary dynamic parameters. Flexibility is limited only by the complexity of the formula allowed by the language and by the comprehensiveness of the dynamic data that is made accessible. It must be pointed out, however, that this increase in flexibility is accompanied by an increase in system complexity—it is now necessary to know the programming syntax that can be used to prepare a formula and to access dynamic data.

It is obvious that programmable simulation systems and simulation programming languages can be very powerful, but at the same time they can be very complex and require a stronger commitment for their mastery. Simple systems can be learned very easily, but are limited to the capabilities that are built into the tool. Certain object-oriented concepts can be applied to the design of systems with the purpose of hiding complexity to the novice without removing the programming capabilities that give flexibility to the advanced user. It is very difficult, however, to provide flexibility and extreme simplicity in the same tool. The degree to which this is achieved is a measure of how good the system design is and determines the long-term success of the simulation program in practice.

Simulation System Features

Features are yet another dimension of simulation systems that affect convenience and ease of use. Simulation system features do not depend on the issues described above, such as programmability (flexibility) or simulation modeling strategy (the suitability for modeling certain types of systems). Instead, features include capabilities such as graphical user interfaces, interactive debugging and tracing features, production of presentation quality reports with tables and charts, and animation. In the case of a general-purpose or simulation programming language, features are a characteristic of the specific implementation and not of the language itself. Different vendors of the same ANSI C++ language (Stroustrup 1998), for example, offer different sets of features in their compilers. Although such features may not be part of the underlying language, they are an integral part of the system and determine its acceptance and long-term use in practice. Thus, construction simulation tools should not only be based on activity scanning and provide for programmability, but must also incorporate a rich set of powerful features, such as a graphical interface, a powerful model debugger, and animation.

GENERAL-PURPOSE SIMULATION TOOLS FOR CONSTRUCTION

ACDs are extremely convenient for construction operations modeling. In practice, however, simulation models must be implemented using actual systems with specific flexibility and feature sets. For many years, the available ACD-based simulation systems had been limited to nonprogrammable tools that were adequate and extensively used for instructional purposes and for modeling systems that were not too complex or detailed. Advanced programmable AS systems were developed primarily in England during the 1960s and 1970s and were proprietary and practically unknown in the United States. Perhaps the best example is HOCUS, a system with significant potential for modeling cyclic construction operations (Halpin 1973).

In contrast, advanced and feature-rich PI simulation languages for modeling manufacturing and service-oriented systems have been widely available since the 1960s. These PI tools have also been used to model various relatively complex construction operations despite the significant effort required to develop appropriate simulation models (Ashley 1980). In the past, these choices were necessary because complex construction processes require a flexibility that was not available in ACD-based tools. Thus, it became common for construction researchers to conceptualize, reason, and explain complex models using ACDs, but to implement them using a PI-based language. A recent example is resource-based modeling (RBM), where ACDs were used to explain the concept and SLAM II was used for the implementation (Shi and AbouRizk 1997). The authors explained that the need to convert the CYCLONE model (an ACD representation) to an equivalent SLAM II simulation model (using a PI-based language) was necessitated by issues of “flexibility.”

Fortunately, this no longer has to be the case. In the remainder of this section we will examine the development of construction simulation tools and conclude that we have at last reached the point where the variety of available AS-based simulation tools for construction is as broad as that for other industries.

Petri Nets

A class of AS-based discrete-event simulation modeling systems are founded on Petri Net concepts. A Petri Net is a formal graphical representation for the analysis of systems with concurrency (Petri 1966). Petri Nets are supported by a general theory of discrete parallel systems that generalizes the theory of automata in a way that allows the expression of concurrently occurring events. Petri Net theory provides formal methods of analysis to determine whether a specific configuration of a system can be reached from another configuration, a system is reversible, a complex system can be simplified, two different systems are functionally equivalent, and many other similar analyses (Jensen 1992).

Since the late 1980s, Petri Nets have also been used for discrete-event simulation. The nodes in a Petri Net alternate between transitions and places. Transitions are typically shown as bars, but are also often shown as squares or rectangles. Places are shown as circles. These nodes are connected with directional edges (links). Tokens initially reside in places. In the basic case, a transition fires when each place preceding it contains at least one token. Upon firing, the transition removes tokens from preceding places and deposits tokens in succeeding places. There is often a holding period that indicates a delay between the time of transition firing (removal of tokens from preceding places) and the time at which tokens are deposited in succeeding places. It must be pointed out, however, that the use of a holding period to support the passage of time robs the methodology of its power because it is incompatible with most formal Petri Net analyses (Jensen 1992).

From this discussion, it is clear that Petri Nets are practically identical to ACDs in functionality and appearance when used to perform discrete-event simulation: A place is analogous to a queue, a transition to an activity, an edge to an arc, a token to a resource, and the holding period of a transition to the duration of an activity. Petri Nets are in fact used to teach three-phase AS concepts in computer science programs (Schruben 1995). In addition, several papers on the relationships between Petri Nets and three-phase AS have been published (Lin and Lee 1993).

Given that for all practical purposes Petri Nets are indeed ACD tools, they show potential for use in the simulation of construction operations. Recent literature already shows uses of Petri Nets for construction modeling (Damrianant and

Wakefield 1997; Sawhney 1997). Numerous Petri Net-based simulation tools exist with different capabilities and features. When used to model construction operations, however, Petri Nets exhibit certain important limitations. One of them is the inability to represent directly bulk materials, such as soil or aggregate, that may exist or be transferred in measurable continuous quantities. Thus, resource quantities expressed as real numbers (i.e., nonunitary) must be modeled by manipulating data attached to discrete tokens. Another limitation is that the rules for configuring Petri Nets are strictly the same as those for pure ACDs and do not allow for the explicit identification of bound transitions (activities) and the elimination of the corresponding superfluous places (queues). These extensions to ACDs have been adopted by simulation tools designed specifically for construction and have proven to be very convenient. Lastly, the language and formal nature of Petri Nets give them a complexity that is not matched with a corresponding increase in flexibility for those interested only in discrete-event simulation.

Systems Designed Specifically for Construction

Several general-purpose simulation systems have been designed specifically for construction operations. They are based on ACDs with extensions that make them more suitable for modeling construction operations. Although several systems are briefly mentioned here, the emphasis is on CYCLONE (Halpin and Riggs 1992) and STROBOSCOPE (Martinez 1996) because they exhibit the least overlap in the flexibility/simplicity spectrum that characterizes general-purpose simulation tools.

CYCLONE

CYCLONE (Halpin and Woodhead 1976) is the oldest and most used general-purpose simulation tool designed specifically for construction. Although Halpin (1973) described the CYCLONE logic in terms of GERT networks (Pritsker and Happ 1966), the flowcharts that describe the CYCLONE simulation logic in (Halpin and Riggs 1992) support the notion that CYCLONE is an AS where the ACD is the simulation model itself and not just a blueprint for a simulation program. There have been at least four different implementations of CYCLONE: Mainframe CYCLONE (Halpin and Woodhead 1976), Insight (Kalk 1980), UM-CYCLONE (Ioannou 1989), and Micro-CYCLONE (Halpin 1990). Numerous capabilities and features have been built into the various implementations of CYCLONE (Dabbas and Halpin 1982; Bernold and Halpin 1984; AbouRizk and Halpin 1990; Lutz et al. 1994; Huang and Halpin 1994). CYCLONE has also been used to model many construction operations including concrete batch plants (Lluch and Halpin 1982) and tunneling (Touran and Asai 1987).

CYCLONE ACDs enhance pure ACDs with two different types of extensions: (1) Conceptual; and (2) functional. "Conceptual" extensions allow for the explicit classification of activities as conditional (called COMBI and shown by squares

with a slash on the top-left corner) or bound (called NORMAL and shown as simple squares) and for the elimination of superfluous queues. These extensions make conceptual CYCLONE ACDs more compact, appropriate, and appealing for construction operations. Consider the conceptual CYCLONE ACD for the simple scraper and pusher operation shown in Fig. 5. It uses four queues and four arcs less than the pure ACD of Fig. 1. In addition, it makes sequences of chained activities explicit. It is, for example, quite clear that the activities *PushLoad*, *Haul*, *DumpAndSpread*, and *Return* follow each other.

"Functional" extensions enable the CYCLONE ACD to be a complete and unambiguous representation of a simulation model. These extensions include additional functions and nodes (GENERATE, CONSOLIDATE, COUNTER), probabilistic branching for activities followed by more than one other activity, the assumption that all resource flows and requirements are unitary, and the assumption that all resource units are indistinguishable and interchangeable. With these enhancements and assumptions it is possible to create a functional CYCLONE ACD that represents a simulation model with all its details.

To illustrate the power and specificity of CYCLONE's ACD representation consider the scraper and pusher operation with side trimming, shown conceptually in the ACD of Fig. 4, with the following specific details.

1. Each scraper has a capacity of 20 m³.
2. Every 20 push-loads generate the need for one side-trimming pass.
3. After push-loading a scraper, the pusher moves toward the side to trim if there are at least five passes worth of trimming to do.
4. The pusher remains trimming at the side until all required trimming passes are complete. The pusher then moves to the push-loading area.

The operation can be entirely represented with the functional CYCLONE ACD of Fig. 6. All nodes in this network are numbered to provide a way to refer to them in a shorthand manner. The actual node numbers are meaningful only for COMBI activities—those with a lower number have priority over those with a higher number. Nodes 3, 10, and 13 have been added to what would otherwise be a conceptual ACD. Node 3 is a consolidator that holds the resources released by Activity *PushLoad* until 100 are accumulated and then releases one resource to Queue 13 and another resource to Queue 11. The GEN 5 for Queue 11, *SideToTrim*, multiplies the incoming resource by five to allow Activity *TrimSide* to take place five times (the GEN function generates five resources for each one coming in). Because Activity *ToSide* has priority over Activity *BackTrack*, the presence of a resource in Queue 13 allows Activity *ToSide* to take hold of the resource in Queue *Rdy-ToBkTrack* before Activity *BackTrack* does. Similarly, the higher priority of Activity *TrimSide* over Activity *ToPushArea* keeps the pusher trimming the sides before moving it to the push-loading area. The GEN 20 at Queue 15 multiplies each

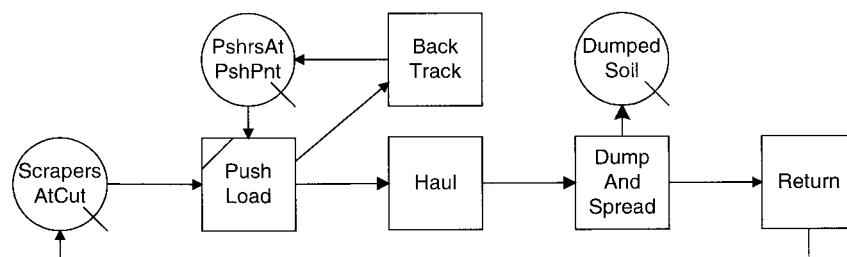


FIG. 5. CYCLONE Conceptual ACD for Simple Scraper and Pusher Operation

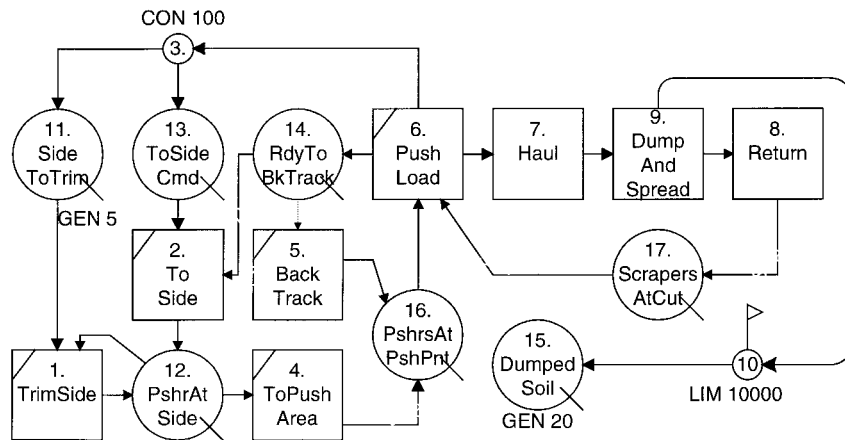


FIG. 6. CYCLONE Functional ACD for Pusher and Scraper Operation with Side Trimming

resource by 20 to convert scraper-loads to cubic meters of soil. The limit of 10,000 on Node 10 indicates that the simulation should run until 10,000 scraper-loads have been dumped.

Fig. 6 does not show the durations of the different activities, the number of initial resources in each queue, and the maximum time limit for the simulation. These details have been omitted for clarity but could have been included to make the network drawing and its annotations a complete representation of the model. For example, the duration of an activity could be defined by specifying a probability distribution and the numerical values of its parameters.

The fact that a CYCLONE model can be completely described by a functional ACD is responsible for both its advantages and limitations. On the positive side, it keeps the modeling methodology simple and easy to learn. It also facilitates communication because everything is contained in the network. The limitations are that many simplifying assumptions must be made when modeling operations that include complex logic, multiple resources with different properties, or nonstationary parameters based on models that have not been built into the CYCLONE implementation. Some examples of these simplifying assumptions and their consequences as well as a comparison between CYCLONE and SLAM II can be found in Gonzalez-Quevedo et al. (1993) and Gonzalez-Quevedo (1995). CYCLONE and SLAM II are two simulation tools that differ in both simulation strategy and programmability. As expected, the writers concluded that CYCLONE is simpler and more natural for construction operations and that SLAM II is more powerful. The simplicity and natural appeal for construction found in CYCLONE is primarily due to the use of ACDs as a means of representation. The power and flexibility found in SLAM II is primarily due to the fact that it is a mature simulation programming language.

CYCLONE has inspired the development of other simulation systems (such as those described below) and its continued widespread use has helped make construction simulation modeling popular.

RESQUE

RESQUE (Chang 1986) incorporates CYCLONE's conceptual and functional extensions, but the model is not limited to the information conveyed by the network. A RESQUE model also includes an overlay that defines resource distinctions and increases simulation control. The overlay provides it with significant flexibility insofar as recognizing distinctions among resources that flow through the same path. For example, the time to load a truck can be sensitive to its type (size) and an activity may be prevented from starting unless the contents of a queue exceed a particular amount.

COOPS

COOPS (Liu 1991) is an object-oriented system that enhances CYCLONE's conceptual and functional extensions with some relaxed node precedence rules. COOPS models are defined via a graphical user interface where all resources are treated as individually identifiable objects to provide statistics from the point of view of each individual resource. These are in addition to the traditional statistics that reflect the point of view of the activities performed or the queues visited. In addition, COOPS allows for the generation and consolidation of resources at links and uses calendars that can be used to preempt activities during work breaks.

CIPROS

CIPROS (Odeh 1992) is both a process level and project level planning tool. It contains an expandable knowledge base of construction techniques and methods, and makes ample use of a hierarchical object-oriented representation for resources and their properties. The resource characterization capabilities in CIPROS go beyond those in RESQUE to allow multiple real properties for resources as well as more complex resource selection schemes. CIPROS also integrates process-level and project-level planning by representing activities through process networks, all of which can use a common resource pool.

STEPS

STEPS (McCahill and Bernold 1993) is a general-purpose system based on pure ACDs that was developed for the U.S. Navy. STEPS supports the notion of different resource sizes in the same queue.

STROBOSCOPE

STROBOSCOPE is a general-purpose simulation programming language with direct support for three-phase AS and ACDs. STROBOSCOPE ACDs represent simulation models only at the conceptual level—the details of a model are specified with other mechanisms that rely on programmability. At the conceptual level, the elements used in a STROBOSCOPE ACD are a superset of those in CYCLONE (e.g., STROBOSCOPE allows for the explicit identification of bound activities with the elimination of the corresponding superfluous queues). In addition, STROBOSCOPE introduces five new nodes and four special types of links of conceptual significance. STROBOSCOPE models, however, do not rely on functional CYCLONE elements (e.g., generate, consolidate, counter) and are not subject to any of the simplifying assumptions found in functional CYCLONE models (e.g., re-

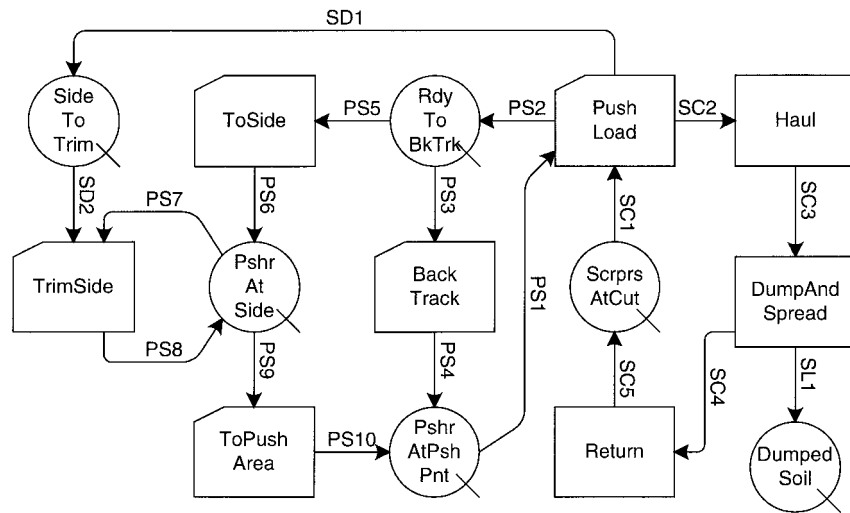


FIG. 7. STROBOSCOPE ACD for Pusher and Scraper Operation with Side Trimming

TABLE 2. STROBOSCOPE Method Redefinitions for Simple Scraper and Pusher Operation with Side Trimming

Element (1)	Method (2)	Redefinition (3)
SD1	Release amount	1
ToSide	Priority	SideTo Trim.CurCount-100
TrimSide	Priority	10
SD2	Draw amount	20
SL1	Release amount	20

sources of the same type can be distinguished from one another and each can have individual properties). Each modeling element in a STROBOSCOPE ACD has methods (attributes) that define the element's behavior. Each method (attribute) has a default implementation that reflects the element's most common behavior, but which can be redefined to achieve the desired effect. Thus, the functional details of a model are mostly specified by redefining these methods. The code used to redefine a method can be as simple as a number, but can also consist of complex expressions that call functions, that access the dynamic state of the model, and that manipulate arrays, statistics collection objects, or files.

To illustrate the development of a STROBOSCOPE simulation model consider the pusher and scraper operation with side trimming described conceptually by the ACD of Fig. 4 and shown as a STROBOSCOPE ACD in Fig. 7. The two main differences between these representations are the elimination of superfluous queues (and the explicit identification of bound activities) and the use of names to identify links in the STROBOSCOPE version. Link names are used to redefine link-related methods and to access link-related dynamic data at run time. By convention, the names used for links are composed of a two-letter abbreviation of the type of resource that flows through and of a number that gives an idea of flow sequence. In this case, the PS links are for pushers, the SC links for scrapers, the SD links for side, and the SL link for soil.

To define a functional simulation model for this operation it is also necessary to establish priority rules for the Activities ToSide, BackTrack, TrimSide, and ToPushArea. For an earth-moving operation with the same specific details as those used to build the functional CYCLONE ACD of Fig. 6, it is necessary to redefine several modeling element methods as shown in Table 2. Except for activity durations, which for brevity are omitted here, it is not necessary to redefine any other element method because the default behavior is exactly what is needed. These redefinitions translate into plain English as follows:

- After push-loading create one unit of "side" for trimming. (In the absence of this statement zero units of side would be created because none are acquired for push-loading.)
- The priority of Activity ToSide depends on the number of side units in need of trimming (SideToTrim.CurCount). This priority becomes increasingly greater than normal (i.e., >zero) as the number increases past 100, but is below normal when the number is <100. (By default the priority of all activities is zero; this "tie" is broken by the order in which the activities are defined.)
- The priority of Activity TrimSide is always greater than normal.
- Activity TrimSide consumes 20 side units every time it takes place. (By default, only 1 unit of side would be consumed.)
- After dumping and spreading, create 20 units of soil. (In the absence of this statement Activity DumpAndSpread would release zero units of soil because it did not receive any.)

Although used for a simple means in this case, the dynamic and intelligent priority of the ToSide activity gives a glimpse of the flexibility that comes with programmability. This flexibility, however, comes at a price. To use the system effectively it is necessary to know that appending ".CurCount" to the name of a queue creates a dynamic variable that returns the queue's current content at run time. It is also necessary to understand how a model will behave by default, and how to alter that default behavior to achieve the desired results.

The necessary redefinitions for a slightly more complex and detailed variation of the scraper and pusher operation, based on the same STROBOSCOPE ACD, are shown in Table 3.

TABLE 3. STROBOSCOPE Method Redefinitions for More Detailed and Complex Scraper and Pusher Operation with Side Trimming

Element (1)	Method (2)	Redefinition (3)
SD1	Release amount	Normal[1,0.05]
SD2	Draw amount	Normal[20,1.5]
ToPushArea	Priority	ScrprsAtCut.CurCount>4 ? 20:0
BackTrack	Priority	ScrprsAtCut.CurCount>3 ? 20:0
BackTrack	Duration	0.25+0.4*(SimTime-PushLoad.LastStart)
Haul	Duration	(800+2*PushLoad.TotInst)/Pert[25,40,45]/16.7

The functional details specified in this table for the more complex example are as follows.

- The amount of side in need of trimming generated by a push-load is not constant but rather follows a normal probability distribution.
- The amount of side that is trimmed in one pass is also not constant and follows another normal probability distribution.
- Backtracking (*BackTrack*) and returning the pusher from the side to the push area (*ToPushArea*) become high-priority activities if the number of scrapers at the cut is high (i.e., >3 and 4, respectively).
- The duration of Activity *BackTrack* is a linear function of the time it took to push-load. The previous push loading time is the difference between the current time (*SimTime*) and the time at which the last push load started (*PushLoad.LastStart*).
- The duration of Activity *Haul* is based on an initial distance of 800 m that is incremented by 2 m after each scraper trip (*PushLoad.TotInst* returns the total number of times that *PushLoad* has taken place). It also depends on the scraper's average speed (in kilometers per hour) as given by a Pert distribution with $P_0 = 25$, Mode = 40, and $P_{100} = 45$.

These example redefinitions were selected to illustrate some of the issues that are easily handled through programmability, but otherwise pose significant difficulties. They include probabilistic resource production and consumption; dynamic activity priorities based on the state of the system; dependence of the duration of one activity on the duration of another, or on any other dynamic system characteristic (such as the constantly changing haul distance); and the capability to embed a sampled random variate in a more complex expression. The inclusion of each one of these capabilities in a nonprogrammable system is a nontrivial undertaking that requires significant independent research.

For the sake of brevity, issues relating to programmability were illustrated in a context that exists in every general-purpose simulation system based on ACDs (e.g., priorities and durations). STROBOSCOPE makes available many other redefinable methods that allow tailoring the behavior of modeling elements very precisely. It also includes programmable modeling concepts that support detailed and complex operations. These include bulk resources; uniquely identifiable discrete resources that can carry information (characterized resources); organization of resources in containment hierarchies of unlimited depth (compound resources); intelligent and dynamic resource routing (forks and dynaforks); non-ACD-derived control of activity start-ups (semaphores); and advanced experimental control and random number stream management for the implementation of variance reduction techniques, sensitivity analysis, and automated replications.

The STROBOSCOPE language also provides facilities for writing code to manipulate variables, arrays, statistics collection objects, files, and the like, at the occurrence of specific events attached to modeling elements during simulation (e.g., write formatted output to a disk file every time a scraper flows through Link SC2). Programming capabilities of another nature include the ability to extend the language with dynamic link libraries written in high-level programming languages such as C++, and the ability to integrate STROBOSCOPE as a component in a multitool decision support system by using its object linking and embedding automation interface.

As with any programming language, it is impossible to mention all aspects of STROBOSCOPE within the limits of this short description. Table 4, however, provides some general data about the language.

TABLE 4. STROBOSCOPE Language Facts

Fact (1)	Number (2)
Conceptual ACD elements (queue, combi, normal, link, dynafork, assembler, etc.)	13
Element methods that can be redefined (duration, priority, strength, draw order, draw when, etc.)	19
Types of dynamically accessible modeling element data (<i>SimTime</i> , <i>queue.AveWait</i> , <i>activity.TotInst</i> , etc.)	111
Events for performing actions at simulation runtime (On Start, Before End, On Release, etc.)	10
Types of statistics collection objects (collectors, weighted collectors, time-weighted collectors, etc.)	10
Probability distributions (Rnd[], Normal[m,s], Beta[a,b], Gamma[a,b], Pert[a,M,b], etc.)	11
Functions (Abs[val], Ln[val], Sin[val], Confidence[sd,lv,n], etc.)	134
Statements (CHARTYPE, SIMULATEUNTIL, WHILE, REPORT, etc.)	86
Operators (! - ^ / * - + >= < < != == & ?: etc.)	20
Random number streams	23,000

STROBOSCOPE has been used to model tunneling operations with variance reduction techniques (Ioannou and Martinez 1996b), lean construction processes (Tommelein 1998), paving operations (Harmelink and Bernal 1998), optimum load-growth effects and variable haul distances in pusher and scraper operations (Martinez et al. 1994), quarry operations (Martinez and Ioannou 1995), impacts of changes in construction work (Cor 1998), and concrete block manufacturing (Sangarajakul 1998) among others. In addition, STROBOSCOPE has been used to create animations of construction operations using PROOF (Ioannou and Martinez 1996a,c,d), to build project-level modeling tools (Wang 1996; Martinez and Ioannou 1996) and special-purpose simulators (Martinez 1997, 1998).

CONCLUSIONS

The essential characteristics of discrete-event simulation systems are their application breadth, modeling paradigm, and flexibility. Application breadth determines whether a system is designed for general purposes or for the modeling of operations within a specific and narrow domain. The modeling paradigm (simulation strategy) used by a general-purpose tool gives it its flavor and makes it naturally suitable for modeling certain types of systems. The main strategies in use today are PI, AS, and ES. ES is often used to enhance PI and AS. The combination of AS and ES is typically called the three-phase approach. Although systems based on any strategy can be used to model construction operations, it is most convenient to represent them using ACDs, which are networks that naturally describe three-phase AS models. The flexibility of a tool is independent of its strategy and application breadth. In general, simple tools are very easy to learn and use, but are limited in their modeling capabilities. More advanced tools typically involve programming and thus require a stronger commitment for their mastery, but are capable of modeling almost any system in detail regardless of its complexity.

Several general-purpose discrete-event systems have been designed specifically for modeling construction operations. These systems are all based on some form of ACDs and use the AS or three-phase AS approach. Two of these systems, CYCLONE and STROBOSCOPE, are representative of the range of modeling tools that are now available for use in the analysis of construction operations. CYCLONE is well-established, widely used, and simple system that is easy to learn and effective for modeling many simple construction operations. STROBOSCOPE is a programmable and extensible simulation system designed for modeling complex construction

operations in detail and for the development of special-purpose simulation tools. These systems indicate clearly that three-phase AS is the wave of the future for construction simulation.

ACKNOWLEDGMENTS

The writers thank the National Science Foundation (Grants CMS-9415105 and CMS-9733267) for supporting portions of the work presented here. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

APPENDIX. REFERENCES

- AbouRizk, S. M., and Halpin, D. W. (1990). "Probabilistic simulation studies for repetitive construction processes." *J. Constr. Engrg. and Mgmt.*, ASCE, 116(4), 575–594.
- Ashley, D. (1980). "Simulation of repetitive-unit construction." *J. Constr. Div.*, ASCE, 106(2), 185–194.
- Balci, O. (1988). "The implementation of four conceptual frameworks for simulation modeling in high-level languages." *Proc., 1988 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 287–295.
- Bernold, L. E., and Halpin, D. W. (1984). "Microcomputer cost optimization of earth moving operation." *Proc., 4th Int. Symp. on Organization and Mgmt. of Constr.*, Vol. 11.
- Birtwistle, G., Lumow, G., Unger, B., and Lucker, P. (1985). "Process style packages for discrete event modeling: Experience from the transaction, activity and event approaches." *Trans. Soc. for Comp. Simulation*, 2(1), 27–56.
- Buxton, J. N., and Laski, J. G. (1962). "Control and simulation language." *Comp. J.*, 5, Swindon, U.K., 194–199.
- Chang, D. Y. (1986). "RESQUE: A resource based simulation system for construction process planning." PhD dissertation, University of Michigan, Ann Arbor, Mich.
- Cor, H. (1998). "Using simulation to quantify the impacts of changes in construction work," MS thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va.
- Dabbas, M. A. A., and Halpin, D. W. (1982). "Integrated project and process management." *J. Constr. Engrg. and Mgmt.*, ASCE, 108(3), 361–374.
- Damrianant, J., and Wakefield, R. R. (1997). "A petri-net based system for modeling and computer simulation of construction operations." *Proc., 4th Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 190–197.
- Evans, J. B. (1989). *Structures of discrete event simulation*. Wiley, New York.
- Gonzalez-Quevedo, A., AbouRizk, S. M., Iseley, D. T., and Halpin, D. W. (1993). "Comparison of two simulation methodologies in construction." *J. Constr. Engrg. and Mgmt.*, ASCE, 119(3), 573–589.
- Gonzalez-Quevedo, A. (1995). "Sensitivity analysis of construction simulation." *Proc., 2nd Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 1427–1434.
- Halpin, D. W. (1973). "An investigation of the use of simulation networks for modeling construction operations," PhD dissertation, Dept. of Civ. Engrg., University of Illinois at Urbana-Champaign, Ill.
- Halpin, D. W. (1976). "CYCLONE—Methods for modeling job site processes." *J. Constr. Div.*, ASCE, 103(3), 489–499.
- Halpin, D. W. (1990). *Micro-CYCLONE user's manual*. Dept. of Civ. Engrg., Purdue University, West Lafayette, Ind.
- Halpin, D. W., and Riggs, S. (1992). *Design of construction and process operations*. Wiley, New York.
- Halpin, D. W., and Woodhead, R. (1976). *Design of construction and process operations*. Wiley, New York.
- Harmelink, D. J., and Bernal, M. A. (1998). "Simulating haul durations for linear scheduling." *Proc., 1998 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 1607–1613.
- Hills, P. R. (1971). *HOCUS*. P. E. Group, Egham, Surrey, England, U.K.
- Hills, P. R. (1973). "An introduction to simulation using simula." *Publ. No. S 55*, Norwegian Computing Center, Oslo.
- Hooper, J. W. (1986). "Strategy related characteristics of discrete-event languages and models." *Simulation*, 46(4), 153–159.
- Hooper, J. W., and Reilly, K. D. (1982). "An algorithmic analysis of simulation strategies." *Int. J. Comp. and Information Sci.*, 11(2), 101–122.
- Huang, R. Y., and Halpin, D. W. (1994). "Visual construction operations simulation—The DISCO approach." *J. Microcomp. in Civ. Engrg.*, 9, 175–184.
- Ioannou, P. G. (1989). "UM-CYCLONE reference manual." *Tech. Rep. UMCE-89-11*, Dept. of Civ. Engrg., University of Michigan, Ann Arbor, Mich.
- Ioannou, P. G., and Martinez, J. C. (1996a). "Animation of complex construction simulation models." *Proc., 3rd Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 620–626.
- Ioannou, P. G., and Martinez, J. C. (1996b). "Comparison of construction alternatives using matched simulation experiments." *J. Constr. Engrg. and Mgmt.*, ASCE, 122(3), 231–241.
- Ioannou, P. G., and Martinez, J. C. (1996c). "Simulation of complex construction processes." *Proc., 1996 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 1321–1328.
- Ioannou, P. G., and Martinez, J. C. (1996d). "Scaleable simulation models for construction operations." *Proc., 1996 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 1329–1336.
- Jensen, K. (1992). *Coloured petri nets. Basic concepts*, Vol. 1, Springer-Verlag, New York.
- Kalk, A. (1980). "INSIGHT: Interactive simulation of construction operations using graphical techniques." *Tech. Rep. No. 238*, Dept. of Civ. Engrg., Stanford University, Stanford, Calif.
- Lin, J. T., and Lee, C. C. (1993). "A three-phase discrete event simulation with EPNSim graphs." *Simulation*, 60(6), 382–392.
- Liu, L. Y. (1991). "COOPS: Construction object-oriented simulation system," PhD dissertation, University of Michigan, Ann Arbor, Mich.
- Lluch, J., and Halpin, D. W. (1982). "Construction operation and microcomputers." *J. Constr. Div.*, ASCE, 108(1), 129–145.
- Lutz, J. D., Halpin, D. W., and Wilson, J. R. (1994). "Simulation of learning development in repetitive construction." *J. Constr. Engrg. and Mgmt.*, ASCE, 120(4), 753–773.
- Martinez, J. C. (1996). "STROBOSCOPE: State and resource based simulation of construction processes," PhD dissertation, University of Michigan, Ann Arbor, Mich.
- Martinez, J. C. (1997). "Structure of an advanced course in computer applications and simulation in construction." *Proc., 4th Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 206–215.
- Martinez, J. C. (1998). "EarthMover—Simulation tool for earthwork planning." *Proc., 1998 CIB W78 Conf.*, Dept. of Constr. Mgmt. and Economics, Royal Institute of Technology, Stockholm, Sweden.
- Martinez, J. C., and Ioannou, P. G. (1995). "Advantages of the activity scanning approach in the modeling of complex construction processes." *Proc., 1995 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 1024–1031.
- Martinez, J. C., and Ioannou, P. G. (1996). "State-based probabilistic scheduling using STROBOSCOPE's CPM add-on." *Proc., Constr. Congr. V*, ASCE, Reston, Va, 438–445.
- Martinez, J. C., Ioannou, P. G., and Carr, R. I. (1994). "State and resource based construction process simulation." *Proc., 1st Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 177–184.
- McCahill, D. F., and Bernold, L. E. (1993). "Resource-oriented modeling and simulation in construction." *J. Constr. Engrg. and Mgmt.*, ASCE, 119(3), 590–606.
- Mohieldin, Y. A. (1989). "Analysis of construction processes with non-stationary work task durations," PhD dissertation, University of Maryland, Md.
- Odeh, A. M. (1992). "Construction integrated planning and simulation model," PhD dissertation, University of Michigan, Ann Arbor, Mich.
- Olooufa, A. A., and Ikeda, M. (1997). "Library-based simulation modeling in construction." *Proc., 4th Congr. on Computing in Civ. Engrg.*, ASCE, Reston, Va, 198–205.
- Paulson, B., Chan, W., and Koo, C. C. (1987). "Construction operations simulation by microcomputers." *J. Constr. Engrg. and Mgmt.*, ASCE 113(2), 302–314.
- Petri, C. A. (1966). "Communication with automata." *Tech. Rep. RADCR-65-377, 1, 1*, Griffiss Air Force Base.
- Pritsker, A. A. (1986). *Introduction to simulation and Slam II*, 3rd Ed. Halsted Press, Wiley, New York.
- Sagert, P. (1995). "Simulation of construction operations," MS thesis, Chalmers University of Technology, Göteborg, Sweden.
- Sangarayakul, B. (1998). "Use of simulation to analyze block manufacturing methods," MS thesis, Virginia Polytechnic Institute and State University, Blacksburg, Va.
- Sawhney, A. (1997). "Petri net based simulation of construction schedules." *Proc., 1997 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 1111–1118.
- Schriber, T. J. (1990). *An introduction to simulation using GPSS/H*. Wiley, New York.
- Schruben, L. W. (1995). "Building reusable simulators using hierarchical event graphs." *Proc., 1995 Winter Simulation Conf.*, Society for Computer Simulation, San Diego, Calif., 472–475.
- Shi, J., and AbouRizk, S. (1997). "Resource-based modeling for con-

- struction simulation." *J. Constr. Engrg. and Mgmt.*, ASCE, 123(1), 26–33.
- Stroustrup, B. (1998). *The C++ programming language*, 3rd Ed., Addison-Wesley, Reading, Mass.
- Szymankiewics, J., McDonald, J., and Turner, K. (1987). *Solving business problems by simulation*. McGraw-Hill, London.
- Tocher, K. D. (1963). *The art of simulation*. English University Press, London.
- Tocher, K. D. (1964). "Some techniques of model building." *Proc., IBM Scientific Computing Symp. on Simulation Models and Gaming*, IBM, New York, 119–155.
- Tocher, K. D. (1979). "Keynote address." *Proc., 1979 Winter Simulation Conf.*, IEEE, Piscataway, N.J., 641–654.
- Tocher, K. D., and Owen, D. G. (1960). "The automatic programming of simulations." *Proc., IFORS Conf.*, IFORS, Aix-en-Provence, 50–67.
- Tommelein, I. D. (1998). "Pull driven scheduling for pipe-spool installation: Simulation of lean construction technique." *J. Constr. Engrg. and Mgmt.*, ASCE, 124(4), 279–288.
- Touran, A., and Asai, T. (1987). "Simulation of tunneling operations." *J. Constr. Engrg. and Mgmt.*, ASCE, 113(4), 554–568.
- Wang, W. C. (1996). "Model for evaluating networks under correlated uncertainty—NETCOR," PhD dissertation, University of California, Berkeley, Calif.
- Zeigler, B. P. (1976). *Theory of modeling and simulation*. Wiley, New York.