VISUAL MODELING AND SIMULATION TOOLKIT FOR ACTIVITY CYCLE DIAGRAM

Donghun Kang and Byoung K. Choi
Department of Industrial & Systems Engineering
KAIST
335 Gwahak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea
E-mail: donghun.kang@vmslab.kaist.ac.kr

KEYWORDS
Activity Cycle Diagram, Activity Transition Table, Three-phase Rule.

ABSTRACT

A direct use of activity cycle diagram (ACD) model into a simulation execution has a limitation that it does not maximize the power of the widely adopted three-phase rule in the simulation execution of ACD models. This paper presents a key model specification for the simulation execution of the ACD model, named, activity transition table (ATT). The proposed ATT reduces the gap between the ACD (the flow of state change) and the three-phase rule (activity transition) and maximizes the modularity of the three-phase rule. The presented ATT model and ACD model can be implemented and executed with the help of the visual modeling and simulation toolkit.

INTRODUCTION

The activity cycle diagram (ACD) is a method to describe the interactions of objects in a system. It uses the common graphical modeling notation to explain series of activities in real-life diverse circumstances.

The core idea of the ACD was conceived by Tocher to describe the congestion problem at the steel plant in a general framework, called *flow diagram* (Tocher 1960) with the *three-phase rule* (Tocher 1963).

The objects in a system can be classified into two classes: 1) transient object or *entity* that receives the services and leaves the system, 2) resident object or *resource* that serves the entities.

In the ACD, the behavior or lifecycle of an entity or resource in the system is represented by an activity cycle, which alternates the active states with the passive states. The passive state of an entity or resource is called a *queue* in a circle, and the active state is called an *activity* in a rectangle as shown in Figure 1. The arc is used to connect the activity and queue.

The activity represents the interaction between an entity and resource(s), which usually takes a *time delay* to finish it. The token is used to represent the state of the queue and activity. All activity cycles are closed on itself (Carrie 1988).

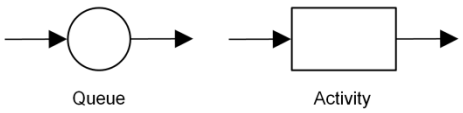


Figure 1: Basic Graphical Notation for the ACD

An ACD model for a single machine system with a setup operator is shown in Figure 2. This model consists of four activity cycles: three for resources of "generator", "machine" and "operator" and one for an entity of "jobs". A job is generated at the interval of t_a time unit by the generator and stored in a queue "B" waiting its processing on a machine. A ready-to-process machine serves a job for t_p time unit if a queue "B" has at least one job and it holds for a moment until the operator is available. The operator sets up the machine for t_s time unit as soon as it is available. Other resources also perform one or more different activities in any sequence or are idle. Here, all activity cycles are closed.

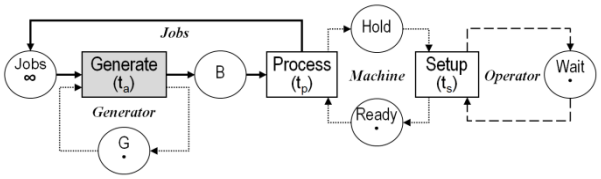


Figure 2: An Example of the ACD - Single Machine System with a Setup Operator

The three-phase rule is also proposed by Tocher (Tocher 1963) to handle the flow of time in the discrete event simulation:

Phase A: Advance the clock to the time of the next (bound-to-occur) event.

Phase B: Terminate any activity bound to end at this time.

Phase C: Initiate any activity whose condition now permits.

The ACD represents the state flow of an entity or resource in a system, while the three-phase rule is based on the event that denotes the change in the state of the model. In phase B, the activities bound to occur at a time are terminated with the release of resources and entities (into output queues), which is called bound or *bound-to-occur (BTO) event*.

Proceedings 24th European Conference on Modelling and Simulation ©ECMS Andrzej Bargiela, Sayed Azam Ali David Crowley, Eugène J.H. Kerckhoffs (Editors)
ISBN: 978-0-9564944-0-5 / ISBN: 978-0-9564944-1-2 (CD)

In phase C, the *conditional events*, which satisfies the beginning condition of the availability of entities and resources, are initiated by acquiring of them (Crookes 1986).

The gap between the ACD and three-phase rule makes difficult to use these well-structured methods for the modeling of the complex system or detailed modeling. For this reason, a key model specification for the execution of ACD models, *activity transition table*, is needed. This describes the dynamics of a system in the viewpoint of the activity transition (of BTO event and conditional event). The activity transition table (ATT) has the one-to-one relationship with the atomistic structure of the three-phase rule. Therefore, it makes the three-phase rule more efficient.

The paper is organized as follows. The second section presents a model specification for the simulation execution of the ACD models, named the activity transition table. The third section presents the three-phase activity scanning algorithm simulating the activity transition table. In the fourth section, the proposed activity transition table and its execution method are realized by the visual modeling and simulation toolkit. At last, conclusions and discussions are provided in the last section.

ACTIVITY TRANSITION TABLE

The three-phase rule has the atomistic structure of advancing time and executing BTO and conditional events. In the simulation execution, the BTO event is handled by the event routine and the conditional event is executed by the activity routine.

The activity routine firstly checks the at-begin condition of an activity, whether all input queues of that activity has at least one token or not. If it is true, the at-begin state-update is fulfilled, which takes one token out of each input queue. Then it schedules a BTO event to occur in a time delay or time duration. The event routine executes the at-end state-update, which adds one token to each output queue.

The phase C of the three-phase rule has an inefficiency of scanning all activity in the ACD model, even though the BTO event has an effect only on the succeeding activities.

The activity transition table (ATT) as a model specification for the simulation execution of the ACD models is a set of activity transitions. Each activity transition has at-begin condition, at-begin state-update, BTO event with the time delay, at-end state-update and influenced activities.

The ATT model can be derived from the following formal definition of the ACD model:

$M = <A, Q, I, O, T, \mu>$, where
A is the finite set of activities,
Q is the finite set of queues,
I: $A \rightarrow \{Q\}$ is the input queues of an activity a_i,
O: $A \rightarrow \{Q\}$ is the output queues of an activity a_i,
T is the time delay function T: $A \rightarrow R+$,
μ is the finite set of tokens for each queue.

The at-begin condition specifies the condition of that every input queue qj of an activity ai should have at least one token, in short, $\mu_j > 0$, for all $q_j \in I(a_i)$. The at-begin state-update is defined as $\mu_j' = \mu_j - 1$, for all $q_j \in I(a_i)$, which decreases the token value of every input queue q_j of an activity a_i by one. The BTO event is scheduled to occur in a time delay of an activity a_i, $T(a_i)$. The at-end state-update is derived by $\mu_j' = \mu_j + 1$, for all $q_j \in O(a_i)$. The influenced activities are defined as a set of activities, $\{a_k \mid q_j \in O(a_i), q_j \in I(a_k)\}$, whose input queue is one of output queues of an activity a_i.

Table 1 shows the ATT model for the single machine system with a setup operator in Figure 2. The queue "Jobs" does not show up in the ATT model, because it is a dummy node used for making the activity cycle closed.

Table 1: ATT Model for the Single Machine System with a Setup Operator

Name	At-begin		BTO Event		At-end	
	Condition	State Update	Time	Name	State Update	Influenced Activities
Generate	G>0	G--	t_a	Generated	G++, B++	Generate, Process
Process	Ready>0 && B>0	Ready--, B--	t_p	Processes	Hold++	Setup
Setup	Hold>0 && Wait>0	Hold--, Wait--	t_s	Setup	Wait++, Ready++	Setup, Process

It is one of the advantages of the ATT model that it can handle some extensions of the ACD model (e.g. arc condition and arc multiplicity) without further extensions of it. This minimizes the modification of the simulation toolkit to cover the extended ACD.

The other advantage of the ATT model is its atomistic structure inherited by the three-phase rule. This enables the automatic code generation with ease.

In addition, the influenced activities of the activity transition make the simulation execution efficient so that the three-phase rule becomes more powerful.

THREE-PHASE ACTIVITY SCANNING ALGORITHM

The three-phase rule for the simulation execution of the ACD models is formally expressed in a three-phase activity scanning algorithm as shown in Figure 3. In this algorithm, two lists are maintained: CAL (candidate activity list) for storing the influenced activities of current activity and FEL (future event list) for storing the bound-to-occur events. CAL is a FIFO (First-In First-Out) queue, while FEL is a priority queue in ascending order of scheduled time.

The three-phase activity scanning algorithm (in short, activity scanning algorithm) starts with the initialization of system state and putting initially ready activities into

CAL. Let us make an example of the single machine system with a setup operator in Figure 2. Tokens for each queue are set to the initial token values. The "Generate" activity is the only activity whose at-begin condition is satisfied at this time. Therefore, it is put into CAL.

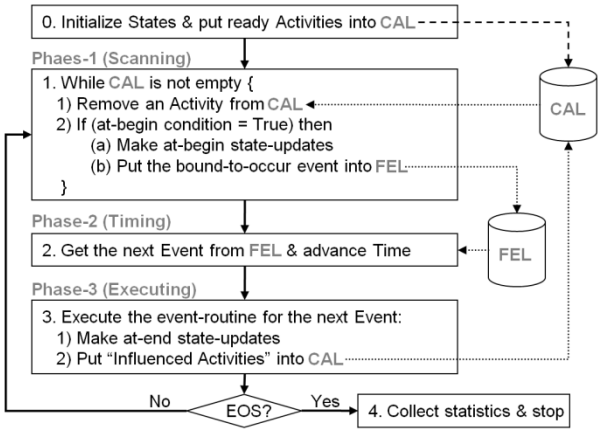


Figure 3: Three-phase Activity Scanning Algorithm

Now at the activity scanning phase of Phase 1, 1) "Generate" activity is removed from CAL, of which the at-begin condition (G>0) is satisfied, 2-a) the token for its input queue "G" are updated, and 2-b) the bound-to-occur event "Generated" is scheduled to occur at time $t_a + 0$ (i.e., stored in FEL). The current system state is:

Current State =
{G=0, B=0, Hold=0, Ready=1, Wait=1}

At the timing phase of Phase 2, the BTO event "Generated" is retrieved from the FEL (the one that has the lowest scheduled time) and the simulation clock is advanced to its scheduled time ($t_a + 0$). Then at the executing phase of Phase 3, the current marking is updated again according to the at-end state-update of the activity "Generate" and the influenced activities ("Generate", "Process") are stored in CAL. At this point, the current marking is:

Current State =
{G=1, B=1, Hold=0, Ready=1, Wait=1}

From now, the above procedure is repeated until it reaches the end-of-simulation condition.

The principle of the three-phase rule has not been changed. The difference between the traditional three-phase rule and proposed activity scanning algorithm is the order of the execution phases. The phase C starts firstly, the phase A is followed, and then phase B is executed. This is because of the influenced activities, which reduces the scanning time of activities in the phase C. The ATT has a more atomistic structure than the three-phase rule does, which is well integrated into the three-phase activity scanning algorithm with more modularity.

VISUAL MODELING AND SIMULATION TOOLKIT

Since Tocher introduced the concept of the ACD, many ACD simulation software tools have been developed in various types of ACD model implementation: 1) automatic code generating simulation software tools such as DRAFT (Mathewson 1985), CAPS (Clementson 1986), and AUTOSIM (Paul and Chew 1987), 2) simulation software tools using simulation language such as CYCLONE (Halpin 1977) and STROBOSCOPE (Martinez and Ioannou 1994), and 3) visual interactive modeling software tools using graphical modeling notations such as EZSTROBE (Martinez 2001) and GroupSim (Araújo et al. 2004).

The automatic code generation and simulation language approaches are not easy to learn and they lack the ability of simple modeling of the complex system. The visual interactive modeling approach, however, provides the graphical modeling notations so that the modeler who is not familiar with the programming can focus on its own role (Pidd and Carvalho 2006).

The modeling and analysis of the complex system still require the customization of the implemented model in the programming language, because the visual interactive modeling software tools only provide simplified output data collection and analysis.

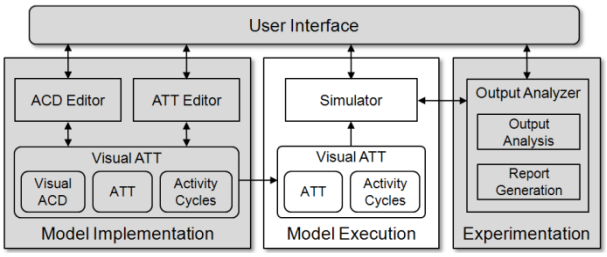


Figure 4: The Visual Modeling and Simulation Toolkit

For this reason, the simulation software tool presented in this paper, as shown in Figure 4, consists of the visual modeling toolkit and simulation toolkit for the support of both visual interactive modeling and custom model implementation. The visual modeling toolkit (in gray) provides graphical modeling notation to implement the ACD model, it also has the ability of modeling the ATT model at the same time and supports output analysis and report generation for the experimentation. The simulation toolkit (in white) is used to execute the simulation and collect the output data during the simulation execution.

Visual Modeling Toolkit

The visual modeling toolkit is developed for the model implementation and experimentation in the simulation model lifecycle. The model implementation can be done with ACD editor to create the ACD model using graphical modeling notation and ATT editor to construct the ATT model simultaneously. Two editors

take roles of view and controller in model-view-controller (MVC) pattern (Buschmann et al. 1996). The visual activity transition table (in short, visual ATT) is the model in the MVC pattern. This maintains the visual ACD (information on queue and activity nodes in a graph), ATT, and activity cycles for entities and resources.

The user can use only one of two editors or both of them. The controller of each editor receives the input from the user and notifies the visual ATT model of the user input, resulting in a change in the model, and then each view of both editors is automatically notified of the change of the visual ATT model.

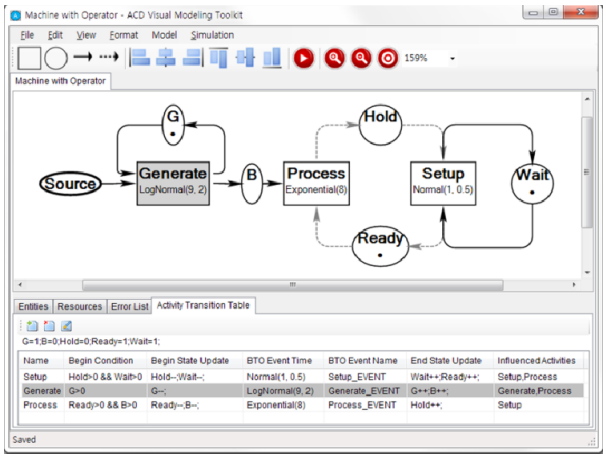


Figure 5: Visual Modeling Toolkit

In Figure 5, the visual modeling toolkit shows the dual view of the single machine system with a setup operator.

The ACD editor located at the middle shows the ACD model using graphical modeling notation: queues in a circle, activities in a rectangle. Inside of the activity node, the name and time delay of an activity is displayed. The activity node in gray represents the initial activity ("Generate" activity), which is ready to begin at the initial state. The queue node shows the name and the initial state of a queue. If the initial state of a queue is more than zero, it is displayed on the queue node with "dot (•)".

The ATT editor located at the last tab of the bottom shows the initial states of all queues and activity transitions. The activity transition can be automatically derived from the ACD model in the ACD editor, or directly inserted or edited by the user using the dialog box for an activity transition.

Prior to the experimentation, the experimental frame should be set to collect the output data and calculate the performance measures by defining the activity cycles of entities and resources. Here, one activity cycle for the "jobs" entity and two activity cycles for the "machine" xand "operator" resource can be made.

Figure 6 shows the output report for resources generated just after the experiment: the utilization of each resource is calculated. The utilization of a machine of single machine system is divided into four states of its activity cycle ("Process" activity, "Hold" queue, "Setup' activity and "Ready" queue).

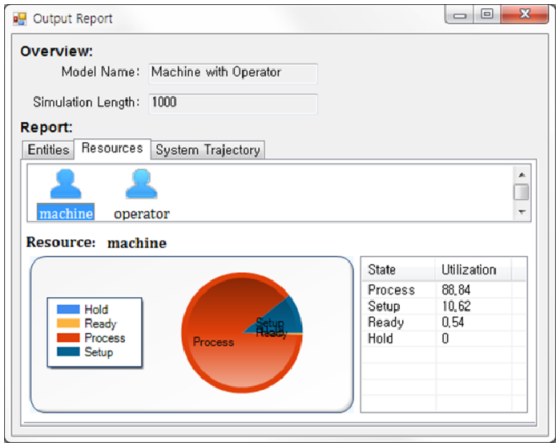


Figure 6: Output Report - Resource View

The visual modeling toolkit stores the visual ATT model into XML document so that it can be exchanged between different simulation toolkits.

Simulation Toolkit

As shown in Figure 7, the simulation toolkit is a set of libraries (model library, simulation library and output data library) for the simulation execution of the ATT model and the output data collection.

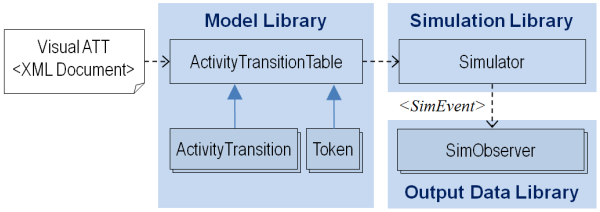


Figure 7: Simulation Toolkit

The model library converts the visual ATT in XML document into the core parts of ATT model in the *ActivityTransitionTable* class with the *ActivityTransition* and *Token* classes. It also supports the modeling of the behavior of the complex system with the inheritance of these classes.

The simulation library consists of the simulator and supporting classes by implementing the three-phase activity scanning algorithm in Figure 3. According to the activity scanning algorithm, 1) in the activity scanning phase, the simulator retrieves an activity transition by invoking the *get-activity* () method on CAL and calls the activity routine of the current activity transition. The activity routine evaluates the at-begin condition of current activity. If it is true, then the at-begin state-update is made to update the token values of

its input queues and the BTO event is scheduled by invoking *schedule-next-event* () method on FEL. This is repeated until CAL becomes empty. 2) In the timing phase, the simulator retrieves the next event by invoking *get-next-event* () method on FEL and advances the simulation clock to the scheduled time of the next event. At last, 3) in the executing phase, the simulator calls the event routine of the next event: the at-end state-update is made to update the token values of output queues and the influenced activities are stored into CAL by invoking *store-activity* () method on CAL.

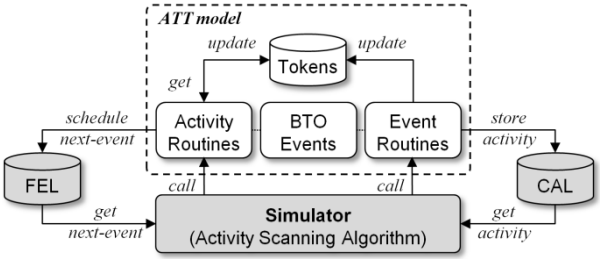


Figure 8: Simulation Library

The output data library is a set of classes and interfaces to support the output data collection, which uses publish-subscribe mechanism of the observer software design pattern (Gamma 1994). Whenever changes are made on the simulation objects such as tokens, CAL and FEL, the simulation events (SimEvent class) are published to the simulator, then these simulation events are distributed to the their subscribers, simulation observers (*SimObserver* class). The simulation observer collects these simulation events to calculate the performance measures.

In our implementation, the simulation events are fired when a BTO event is scheduled or fired or when the token value is changed. Currently, three types of the simulation observer are available: 1) event counter that records the occurrence of a specific simulation event and 2) entity/resource observer that collects any simulation event related to the activity cycle of an entity/resource.

The visual modeling toolkit and simulation toolkit work on Microsoft's .NET Framework (3.5 version) and are implemented with C# programming language. They are available at authors' web site (http://vms.kaist.ac.kr).

CONCLUSION

Proposed in the paper, the ATT is a model specification for the simulation execution of the ACD model. Distinctive features of the proposed ATT includes (1) it reduces the gap between the ACD model and its simulation execution, (2) it covers some extensions of the ACD model without further modification of the simulation toolkit, and (3) it makes the simulation execution more efficient enabling the three-phase rule to become more powerful.

The visual modeling toolkit supports both ACD and ATT modeling views. It helps the ease of ACD modeling with the graphical modeling notation and the automatic generation of ATT model of the XML document. Then, the simulation toolkit uses the ATT model to execute the simulation using the three-phase activity scanning algorithm and supports the output data collection with the publish-subscribe mechanism.

While two symbols of the ACD makes easy to learn and to model, it lacks the modeling power to describe the complex system in details (Hlupic and Paul 1994). For this reason, some researchers (Araúro and Hirata 2004; Kienbaum and Paul 1994; Martinez and Ioannou 1994; Halpin 1977) have proposed many extensions. As an academic research, the generality of the extensions has to be proved with the formal definition and it also needs to prove its real modeling power with its applications to the complex systems, such as automatic material handling system in the factory and general job shop system.

REFERENCES

Araújo, W.F.; C.M. Hirata; and E.T. Yano. 2004. "GroupSim: A Collaborative Environment for Discrete Event Simulation Software Development for the World Wide Web." *SIMULATION*, Vol. 80, No. 6, 257-272.

Araújo, W.F. and C.M. Hirata. 2004. "Translating Activity Cycle Diagrams to Java simulation Programs". In *Proceedings of the 37th Annual Simulation Symposium*. IEEE, Piscataway, N.J., 157-164.

Buschmann F.; R. Meunier; H. Rohnert; P. Sommerlad; M. Stal. 1996, "Model-View-Controller". In *Pattern-oriented Software Architecture: a System of Patterns*. John Wiley & Sons, Chichester, N.Y., 125-144.

Carrie, A. 1988. *Simulation of Manufacturing Systems*. John Wiley & Sons.

Crookes, J.G.; D.W. Balmer; S.T. Chew; and R.J. Paul. 1986. "A Three-Phase Simulation System Written in Pascal." *Journal of the Operational Research Society*, Vol. 37, No. 6, 603-618.

Clementson, A.T. 1986. "Simulating with Activities Using C.A.P.S./E.C.S.L.". In *Proceedings of the 1986 Winter Simulation Conference*, 113-122.

Gamma, E.; R. Johnson; R. Helm; and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Halpin, D.W. 1977. "CYCLONE: Method for Modeling Job Site Processes." *Journal of the Construction Division*. ASCE, Vol. 103, No. 3, 489-499.

Hlupic, V. and R.J. Paul. 1994. "Simulation modelling of flexible manufacturing systems using activity cycle diagrams." *Journal of the Operational Research Society*, Vol. 45, 1011-1023.

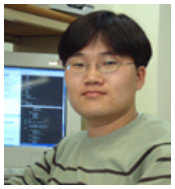
Kienbaum, G. and R.J. Paul. 1994. "H-ACD: Hierarchical Activity Cycle Diagrams for Object-oriented Simulation Modelling". In *Proceedings of the 1994 Winter Simulation Conference*. 600-610.

Martinez, J.C. and P.G. Ioannou. 1994. "General Purpose Simulation with STROBOSCOPE", In *Proceedings of the 1994 Winter Simulation Conference*. 1159-1166.

Martinez, J.C. 2001. "EZStrobe: General-purpose simulation system based on activity cycle diagrams". In *Proceedings of the 2001 Winter Simulation Conference*. 1556-1564.

Mathewson, S.C. 1985. “Simulation Program Generators: Code and Animation on a P.C.” *Journal of the Operational Research Society*, Vol. 36, No. 7, 583-589.
Paul, R.J. and S.T. Chew. 1987. “Simulation Modelling Using an Interactive Simulation Program Generator.” *Journal of the Operational Research Society*, Vol. 38, No. 8, 735-752.
Pidd, M. and A. Carvalho. 2006. “Simulation software: not the same yesterday, today or forever”, *Journal of Simulation*, Vol. 1, No. 1, 7-20.
Tocher, K.D. 1960. “An integrated project for the design and appraisal of mechanical decision-making control systems.” *Operational Research Quarterly*, Vol. 11, No. 1/2, 50-65.
Tocher, K.D. 1963, *The art of simulation*, English Universities Press.

AUTHOR BIOGRAPHIES



Donghun Kang is a PhD candidate student in the Department of Industrial & Systems Engineering at KAIST. He received as BS from KAIST in 2003 in Computer Science, a MS from KAIST in 2005 in Industrial Engineering. His research interests are in the area of system modeling and simulation. His e-mail address is donghun.kang@vmslab.kaist.ac.kr and his web site can be found at http://www.dhkang.org/.

Byoung K. Choi is a professor of the Department of Industrial Engineering at KAIST since 1983. He received a BS from Seoul National University in 1973, a MS from KAIST in 1975, and a Ph.D. from Purdue University in 1982, all in Industrial Engineering. His current research interests are system modeling and simulation, BPMS, simulation-based scheduling, and virtual manufacturing.