Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

CONCEPTUAL SIMULATION MODELING: THE STRUCTURE OF DOMAIN SPECIFIC SIMULATION ENVIRONMENT

Kitti Setavoraphan
Floyd H. Grant

202 W. Boyd St., Room 124
School of Industrial Engineering
University of Oklahoma
Norman, OK 73019, USA

ABSTRACT

This study focuses on the development of a conceptual simulation modeling tool that can be used to structure a domain specific simulation environment. The issues in Software Engineering and Knowledge Engineering such as object-oriented concepts and knowledge representations are addressed to identify and analyze modeling frameworks and patterns of a specific problem domain. Thus, its structural and behavioral characteristics can be conceptualized and described in terms of simulation architecture and context. Moreover, symbols, notations, and diagrams are developed as a communication tool that creates a blueprint to be seen and recognized by both domain experts and simulation developers, which leads to the effectiveness and efficiency in the simulation development of any specific domains.

1 INTRODUCTION

In the past ten years, there have been several panel discussions at, e.g., the Winter Simulation Conferences (Zhou, Son, and Chen 2004; Heavey and Ryan 2006; Robinson 2006a), the OR Society Simulation Workshop (Robinson 2006b, Wang and Brooks 2006), and the BETADE Workshop (Verbraeck and Dahanayake 2002), which acknowledge the use of conceptual modeling (CM) approach and domain specific simulation environment (DSSE) approach as a critical step to improve the quality and efficiency of discrete event simulation research studies/projects. The literature mainly states that good practice of these two approaches significantly reduce communication barriers, organize model structure, shorten project time, and improve simulation development processes (Vreede, Verbraeck, and Eijck 2003; Valentin and Verbraeck 2005; Zhou, Zhang, and Chen 2006). Although their advantages are addressed and supported in the same direction by several simulation studies, CM and DSSE still have so far received little attention from simulation developers because CM is viewed as more of an art than science (Brooks 2006), while DSSE is lack of trust of those (Valentin and Verbraeck 2005).

Numerous articles of, for example, Cyre (1999); Deursen, Klint, and Visser (2000); Pace (2000); Yilmaz and Oren (2004); Valentin and Verbraeck (2005); and Robinson (2006a, 2006b), propose ideas on definitions, requirements, limitations, and methods for the development of CM and DSSE to overcome the struggles in those simulation developers' mind. However, most of them are still reluctant to apply CM and DSSE approach to develop their simulation projects. This is because only a few number of literature demonstrate how to transform and develop those concepts into a standard method/tool that can be used to capture and describe elements required for both CM and DSSE. A research study by Teeuw and van den Berg (1997), for instance, introduces the conceptual framework as developed in their Testbed project by using symbols and notations to describe a system's behaviors, relations, and entities. A conceptual model can also be built by using knowledge representation notations such as semantic/logical graphs, where the nodes represent concepts, and the arcs represent relationships among concepts (Cyre 1999; Zhou, Son, and Chen 2004). Furthermore, a selective review of a number of current modeling methods/tools carried out by Heavey and Ryan (2006) shows that simulation developers have become more aware of using standard methods/tools such as Petri Nets, DEVS, IDEF3, and UML, to develop their own conceptual models. As well, simulation building block terminology is proposed by a research team, BETADE, at Delft University of Technology, The Netherlands, in 2001 to provide a standard methodology for the DSSE development (Verbraeck and Dahanayake 2002), instead of relying on old-fashioned programming. It can be said that the trend of the CM and DSSE research studies is moving forward to acquiring more sophisticated, universal, and user-friendly methods/tools to serve both CM and DSSE requirements effectively and efficiently. However, none of the available methods/tools exists to satisfy this demand.

One of the critical reasons is that both CM and DSSE are viewed from different perspectives that not only isolate them into two distinct disciplines but also eliminate an opportunity for their collaborative modeling and representation formalisms in developing simulation projects. The fact that the foundations of modeling concepts and processes for CM and DSSE are similar allows them to overlap in some aspects (see Valentin and Verbraeck 2002; Verbraeck and Valentin 2002; Vreede, Verbraeck, and Eijck 2003; Zhou, Setavoraphan, and Chen 2005). The concepts developed by CM processes are transformed into the logical and structural components for DSSE, whereas the result of the implementation of those in DSSE becomes a feedback mechanism that provides a better understanding of both the problem domain and the simulation domain – to allow the problem owners to improve their conceptual models for better DSSE (see Figure 1). This iterative CM and DSSE development process is performed until DSSE generates a complete standard set of the specifications and patterns – that can be transformed into basic building blocks. Then these building blocks are composed into a stand-alone simulation template, which is capable of representing systems as a domain specific simulation model for the simulation builders as well as a domain specific conceptual model for the domain experts. Consequently, the simulation template is delivered as the ultimate piece to develop (commercial) simulation software and a knowledge-based simulation system.

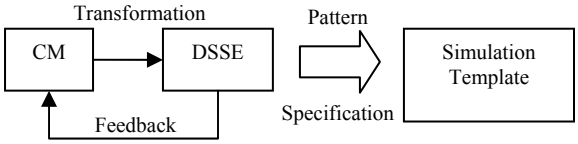

Figure 1: The relationship between CM and DSSE

The main idea of this research study is to specialize CM concepts and techniques to further its potentials in generalizing the behavioral and structural characteristics of a specific problem domain to generate a model that contains processes, elements, controls, and requirements for simulation – that is generally referred to as conceptual simulation modeling (CSM). Thus, the CM approach is determined as the backbone of the development of a CSM tool that can be used to structure DSSE for discreet-event simulation modeling problems. Section 2 briefly describes the key concepts within Software Engineering (SE) and Knowledge Engineering (KE) that baseline the foundations of the CSM development. The concepts are formalized into different layers and representations to construct standard symbols, notations, and diagrams to be used in CSM, which is illustrated in Section 3, including an example for illustration. Finally, conclusions and further research are given in Section 4.

2 KEY CONCEPTS

The simulation development process is a kind of problem-solving process that determines a context, environment, or boundary of a real-world problem domain to be developed and operated. In such the process, CSM plays a critical role as a specialized tool to facilitate the understanding of the problem, support communication between domain experts and simulation developers, and represent the knowledge needed by the simulation system to solve the problem. CSM also takes advantages from the convergent underlying concepts used to develop conceptual models from both SE and KE, which are: first, object-oriented concepts from the discipline SE; and second, knowledge representations (or levels in some literature) from the discipline KE (see more details about the CM methods in Dieste et al. 2001). However, CSM acquires more acknowledged approaches to access, formalize, and handle these concepts to overcome the barriers and drawbacks during constructing and transforming a conceptual simulation model. These are decomposition and composition approaches.

2.1 Decomposition Approach

The significant problem found in applying object-oriented concepts and defining knowledge representations is how to determine and represent the concepts derived from both application knowledge and simulation knowledge (see Zhou, Son, and Chen 2004) at an appropriate abstract level to satisfy the efficiency of CSM. The determination of the level of abstraction is strongly influenced by the objectives of the design or the questions needed to be answered (Benjamin et al. 1993). Nevertheless, no single abstract model is sufficient to be expressed at different levels of precision and to attack specific problems (Booch, Jacobson, and Rumbaugh 1999).

Decomposition is a crucial approach used to handle complexity and represent the behavioral and structural characteristics of the target problem domain at an appropriate level of detail (Zhou, Setavoraphan, and Chen 2005). Moreover, it is the paramount idea of the object-oriented concepts (Meyer 1997), which is used to formalize modeling frameworks for CSM due to its inherent support for abstraction-centric, reusable, and adaptable design (Zhou, Zhang, and Chen 2006). Consisting of abstraction, aggregation, and specialization aspects, the object orientation entrusts a power of decomposition to the simulation developers to capture descriptions at varying abstraction levels and represent all those sub domains into a comprehensive behavioral description. The central idea of decomposition is to breakdown the complexity of a problem domain into less complex sub domains by eliminating irrelevant details and highlighting the important behavioral and structural characteristics (Hofmann 2004); in a meantime, the frames of reference of these sub domains can be extended or mod-

ified to satisfy the objectives of the design (Lee and Wyner 2003). This allows each sub domain to interact with a set of other sub domains for task execution to complete the overall goal of the domain (Davis 2001). For further benefits and criteria of decomposition, see the works by Davis (2001) and Hofmann (2004).

However, to avoid confronting the "more of an art than science" thought during the CSM processes, the constraints of decomposition need to be specified to manage a-kind-of abstraction issue. A proceeding paper (Zhou, Setavoraphan, and Chen 2005) proposes a set of mathematics notations to describe the functions and constraints of two types of decomposition: serial decomposition and parallel decomposition. In the paper, a process-oriented view is used to define a problem domain as a set of sequenced processes in a generic level, which can be decomposed into (multi) sub lower-level processes, controlled by constraints. These constraints are addressed here in narrative descriptions instead of mathematics notations. First, serial decomposition must satisfy the following constraints:

- A top level process must be decomposed into sub processes in order to a serial-sequence order, while each sub process' input and output must be given within its process time.
- The set of sub processes must be a partition of its higher-level process, completely dividing the functionality of the higher-level process.
- Precedence relation is required among the sub processes.
- The attributes defined for the sub processes and the aggregation of these definitions must be consistent with the attributes defined for the decomposed process.
- The input and output external to the set of sub processes must match the original input and output associated with its higher-level process.
- The total process time is a sum of sub process times.

Second, parallel decomposition mostly follows the constraints defined in serial decomposition. The difference is that parallel decomposition requires Boolean logical operators, for example, AND, OR, and XOR, to support the functionalities of logical branching out (e.g., deterministic branching or probabilistic branching) from the predecessor of the original process. These logical operators allow decomposition to create several alternative combinations of causes and effects to extend the consequences of the original process (Bell, Snooke, and Price 2005). As a result of decomposition, the simulation developers are able to capture a set of sequential processes within the domain, corresponding to the simulation requirements to create a conceptual simulation model at the appropriate levels of detail.

2.2 Composition Approach

Another encountered problem is that most of the products (outcomes) from CSM fail to be reused in new simulation development. Reusability of models, modules, or elements is a challenge not only at abstraction level (conceptual simulation models) but also at implementation level (domain specific simulation environments). The failure of capturing and explicitly representing such specifications of constraints, objectives, features, and semantics of components at the conceptual level generates an incompatible framework of those within the domain specific simulation environment, reducing the reusability of model constructs. On the other hand, the incompleteness of encapsulating (modularizing) and inheriting data (e.g., objects and processes) of the model constructs creates the loss of the model functionalities and contexts at the implementation level, affecting the trust of simulation developers in the conceptual simulation models – which results in the distortion of their reusability. Thus, it is a need for an approach to support the model reusability for these two levels.

Composability is described as an approach with a compositional mechanism that provides "the ability to compose models/modules across a variety of application domains, levels or resolution and time scales" (Kasputis and Ng 2000), plus "the capability to select and assemble simulation components in various combinations into simulation systems to satisfy user requirements" (Petty and Weisel 2003). Though, the current capability in composability is limited (Kasputis and Ng 2000) due to the complexity of the selection of components in the context of simulation (Winnell and Ladbrok 2003) – determined as an NP-hard problem, still the simulation developers can apply this approach to design a frame of reference for the possible compositions to increase the possibility of model for reuse in any environment.

In general, there are two types of composability: syntactic composability and semantic composability, used to represent the modeling formalism for the selection of components (Petty and Weisel 2003). First, syntactic composability requires compatible implementation details which include timing mechanisms and interface specifications for all possible compositions. Second, semantic composability requires a meaningful/valid composition. "Both syntactic and semantic composability are necessary for simulation composability" (Bartholet et al. 2004) in terms of the development of the interfaces and the component internals within the defined simulation framework.

In addition, composability can be conducted in two dimensions: horizontal and vertical dimension (Page and Opper 1999). In the horizontal dimension, the components are inter-operated in terms of peer-to-peer integration by justifying a level of modeling abstraction with respect to a set of modeling objectives, which is fundamentally hard to do correctly. On the other hand, composability in the verti-

cal dimension facilitates a level of modeling abstraction through aggregation/disaggregation, which may in turn not provide the best or even a valid solution. It can be seen that the vertical of composability is more flexible to facilitate the composition of decomposed components to create a model corresponding to the specific requirements. Therefore, composability in the vertical dimension is mainly applied in this study to avoid the complexity. Though, it may compensate with the loss of validity.

Butler (1998) identifies three crucial techniques: assembly, extension, and parameterization, as follows:

- Assembly: connecting existing modeling components in possibly unique ways through a common environment.
- Extension: modifying or extending the original functionality of an existing model component through either function override or selective feature activation/deactivation.
- Parameterization: changing parameters which control the operational and behavioral characteristics in an existing model component.

He also states the design requirements for composability to shape the technical and operational approach is his work. Moreover, a number of research studies have been conducted to investigate modeling formalism, context, dependency, and framework for model reuse (Yilmaz and Oren 2004; Spiegel, Reynolds, and Brogan 2005; Sarjoughian and Huang 2005) to improve and facilitate model composability.

The results from these studies reveal not only the techniques of model composability but also the impact of model composability choices in a variety of degrees of model compositions, limitations, and complexity. The idea behind these results shows that the concepts, theories, and techniques of model composability consist of abstraction, hierarchy (aggregation/disaggregation), and encapsulation – that belong to the object-oriented aspects (Sarjoughian and Huang 2005). These aspects are crucially used to develop a framework that provides standardized patterns to define the scopes of design and development of model components and representations for CSM and DSSE. It must be kept in mind that as long as a set of the model components and representations are pattern-based developed within the framework, the reusability of the conceptual simulation models and the model constructs in DSSE is more flexible and more meaningful when conducting a new simulation project. Moreover, it needs to make sure that the composition of the model components and representations must be tested in the level of CSM prior to implement those in DSSE – to avoid the conflicts of functionalities between these two levels.

3 ILLUSTRATION OF A CSM PROTOTYPE

3.1 Background of Study

In the previous section, a focus on the importance of the decomposition and composition approach is given by a means of the application and control to the use of the key concepts: the object orientation and knowledge representation, in the development of a CSM tool. The main reason is that paying less attention in such the management of modeling complexity (levels of detail) and the arrangement of modeling compatibility (levels of selection) results in ineffectiveness and inefficiency of the overall modeling structure and context. Most of the simulation developers know the basic object-oriented concepts described in many publications (e.g., Rumbaugh et al. 1991, Coleman et al. 1993), but a few of them recognize the methods of formalism of these concepts to develop robust and reusable knowledge representations as modeling frameworks for simulation (see Zhou, Zhang, and Chen 2006). It has been found out that there are many generic (standard) methods/tools that are available to support CM (e.g., IDEF3, DEVS, Petri Nets, and UML), but they fail to accomplish transferring concepts and information between application domain and simulation domain, in both directions. As a result, most of the time these methods/tools are simply to create difficulties in the CSM and DSSE construction and translation rather than to achieve the simulation-template's goal.

It can be said that there is a need for a defined simulation modeling framework that facilitates not only the domain conceptualization but also the simulation implementation. A thesis (Setavoraphan 2005) illustrates a CSM tool, called "Simulation Modeling UOB" (SMU), used to formalize concepts into a simulation modeling framework. This tool is developed from the transformation of knowledge representations in a platform of process descriptions derived from IDEF3 method, collaborating with the object-oriented approach. Each instance in SMU employs both process-oriented and component-based view to represent the processes lying within the target problem domain and the simulation modeling elements (e.g., entities, attributes, and functions) satisfying the simulation requirements. Furthermore, it is able to apply the object-oriented features to facilitate modeling decomposition and composition. Having the capability to formalize concepts at different levels of detail and to generate robust and reuse modeling frameworks, SMU has been applied to develop conceptual simulation models for a variety of application domains such as warehousing operations (Setavoraphan 2005) and inland waterway lockage operations (Setavoraphan and Grant 2008). However, the current capability of SMU is limited to deliver a detailed modeling framework that provides both static and dynamic representations required for structuring DSSE.

The original simulation projects developed under the DSSE approach include:

- Electroplating Simulation Program, ESP (Grant and Pritsker 1974) by using a programming language;
- Safeguards Network Analysis Procedure, SNAP (Miner and Grant 1978) by developing a network simulation language;
- Airport Terminal Modeling of Amsterdam Airport Schiphol (Verbraeck and Valentin 2002) and the Robotized Marine Container Terminals (Saanen 2004) by using simulation building blocks.

Accordingly from above, it can be said that every single DSSE development fundamentally consists of static modeling components (e.g., physical layouts) and dynamic modeling components (e.g., entities). These fundamental concepts need to be integrated into the CSM tool for better mapping and transforming concepts prior to develop a simulation modeling framework. A research study by Iba, Matsuzawa, and Aoyama (2004) emphasizes on the Model Driven Development created based on Model Driven Architecture and Executable UML to use high-level modeling languages to enhance the capability of CM in representing the overall behavioral and structural characteristics of a domain, including their interactions, from both static and dynamic views. Their project development supports the idea of improving SMU, the existing CSM tool, by integrating its original concepts with UML to cover its limitations and to be used in this study.

3.2 General Structure

The main purpose of this paper is to deliver a concrete idea that integrates and formalizes the concepts mentioned in the preceding sections by illustrating a CSM prototype temporarily named as "Integrated Simulation Acknowledge Procedure" (ISAP). ISAP is a tool for capturing the concepts in a specific problem domain and transforming them into a set of descriptive processes, static and dynamic modeling components, interactions, and rules/algorithms which are defined within a simulation modeling framework. The framework created by ISAP consists of three layers: the initialization layer (IL), the process layer (PL), and the termination layer (TL) (see Figure 2). First, IL provides initial information about the simulation experiment to be performed (e.g., number of simulation runs, number of attributes/variables, and time to begin/end simulation). Second, PL describes the behavioral and structural characteristics of the problem domain and simulation domain. Third, TL sets the procedures of terminating simulation and printing out a simulation output report. Each of these layers consists of a group of ISAP symbols, notations, and diagrams which are arranged to define and represent modeling structures, elements, and relationships. With the limited space of this paper, only the process layer is discussed.

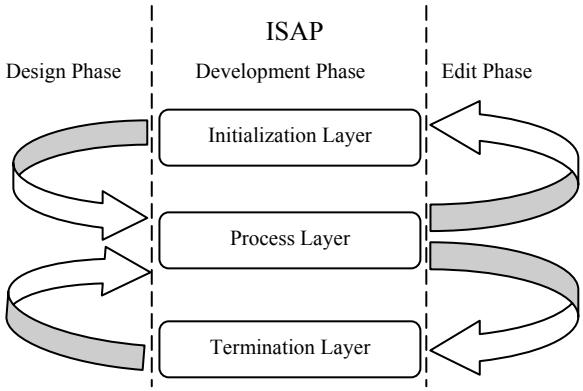

Figure 2: Three layers in ISAP with three phases

The construction of ISAP is based on the modeling and simulation process (Pritsker and O'Reilly 1999) and adapted into three phases: the design phase, the development phase, and the edit phase (also see Figure 2). First, the design phase is to formulate problem and specify model for PL according to the design objectives in IL and TL. Second, the development phase is to build models individually for each layer. Third, the edit phase is to test models and use their feedbacks to correct errors found in these layers, and also this phase needs modification in IL and TL to satisfy the new requirements for PL. Moreover, the construction of PL is divided into two subsystems: static modeling subsystem and dynamic modeling subsystem. Both of them requires the use of symbols, notations, and diagrams for robust and reusable representations. An example of an inventory system of a large discount house (Pritsker and O'Reilly 1999) is used to illustrate the construction of these two sub systems in PL.

3.3 Demonstration

To illustrate these concepts described above, consider a large discount house that is planning to install a periodic review-reorder point inventory system to control its in-house inventory of a particular radio. This system is able to manage backorders in the case where customers demand the radio when it is not in stock. 80 percent will go to another discount house to find it, determined as lost sales, whereas the other 20 percent will be put on the backorder list and wait for the next shipment arrival. The inventory status is reviewed every four weeks to decide if an order should be placed. The company policy is to order up to the stock control level of 72 radios whenever the inventory position, consisting of the radios in stock plus the radios on order minus the radios on backorder, is found to be less than or equal to the reorder point of 18 radios. The procurement lead time requires constantly three weeks.

3.3.1 Static Modeling Subsystem

The first step is to specify the physical characteristics in the target problem domain. It can be seen that the inventory system consists of an actual (in-house) inventory subsystem and a virtual (periodic review-reorder) inventory subsystem. ISAP provides symbols and notations that represent different three static components: BUILD, SPACE, and CROSS. A BUILD component is used to identify a point in a system where some physical objects are moved through or changed their states. A SPACE component is used to identify an area in the system through which physical objects may pass or temporarily stay. A CROSS component is used to identify locations in the system which is the physical objects are engaged with multi cross-domain subsystems. In this example, only BUILD components are used to represent the actual inventory subsystem and the virtual inventory subsystem, where the flows and transition states of e.g., demands and order-signals, take place, shown in Figure 3. Each BUILD component is defined with its identical component label that is connected to its dynamic modeling subsystem containing the logical process flows and parameters needed. The connection is made through "@" and followed by a specified dynamic modeling subsystem label (DMSL). An arrow is used to indicate a precedence of movement that may occur in only one direction between two physical components, which means there exist one or more interchanges or flows of objects and information between the components. One of the obvious benefits of having static (physical) components for CSM is a top-view perspective that shows the core structures and the focused frames of the domain, which can be further developed either as apiece (decomposition) or as a whole (composition) within the defined domain structural boundaries.

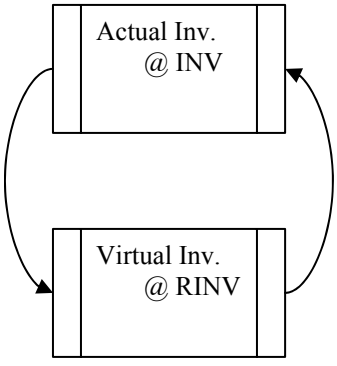

Figure 3: BUILD components for inventory system

3.3.2 Dynamic Modeling Subsystem

The next step is to describe the dynamics of the domain in terms of application knowledge and simulation knowledge, determined as the core of the ISAP development process. Each dynamic modeling subsystem can be view as a document folder that has its own label (DMSL), sub-folder(s) (Ref#), and page number ($ #). Each page is divided into three sections: the SMU section, the relation section, and the sequence-diagram section. The first section follows the major structure described in Setavoraphan (2005), shown in Figure 4, whereas the rest of the sections apply the symbols, notations, and diagrams which are adapted from the UML modeling approach (Booch, Jacobson, and Rumbaugh 1999).

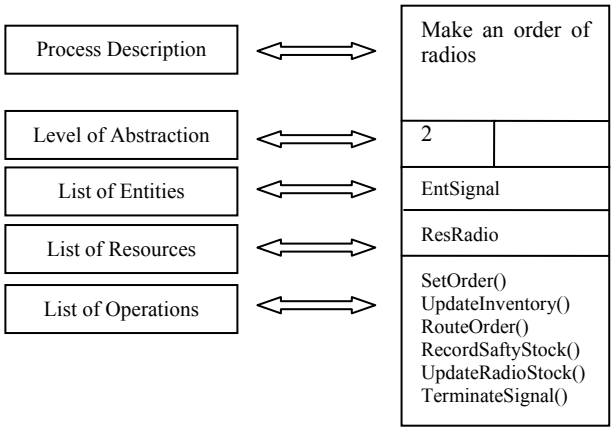

Figure 4: General structure and an SMU example

Each SMU is used to represent as an intimate simulation (block) module that moves the entities through the process or change the entities' transition states; call the resources required for the process; and execute the operations to complete the process. As a module, an SMU can be decomposed into two or more sub SMUs to cover the detailed levels of the process. For example, SMU *Make An Order of Radios* (Fig. 4) can be decomposed into SMU *Prepare An Order* and SMU *Make A Transshipment*, shown in Figure 5. At lower levels of a decomposition, the reference number for level of abstraction of a child SMU (X) consists of three distinct numbers separated by periods. The first number is the last number in the reference number of X's parent SMU. The second number is the number assigned to the particular decomposition of the parent SMU in which X occurs (Note: Numbers are assigned to a set of decompositions and SMUs in order of different creations/points of view). Finally, the third number is an actual X's SMU reference number. The relationship between the parent SMU and child SMUs is determined as a-part-of relationship or aggregation in which SMUs representing the entities, resources, and operations of some processes are associated with an SMU representing the entire assembly of processes. Thus, each decomposition must be taken carefully to avoid the loss of details and the incompleteness of the process. As well, the composition of the existing SMUs into a new SMU requires standard/common pa-

rameters to reduce the invalidity of the model functionalities, which is similar to the methods used in the object-oriented programming. Suppose that the SMUs in Figure 5 are individual SMUs. To compose these two SMUs into one, a crucial requirement is to make sure that they assess the same entities, utilize the same resources, and execute the operations with the same attributes and variables. Also, the flow of entities and operations must be logical sequences.

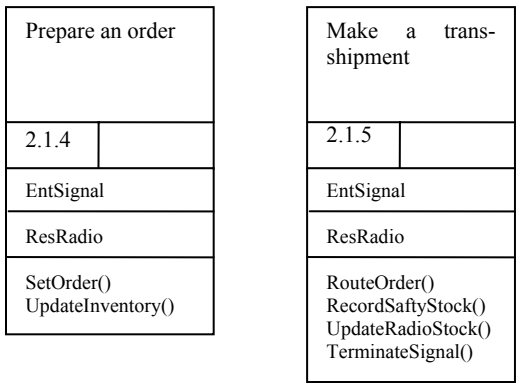

Figure 5: A decomposition of SMU *Make An Order of Radios*

The relation section provides information of the conditions and decisions for branching, preceding, and interacting between two SMUs. Figure 6 gives some examples of notations.

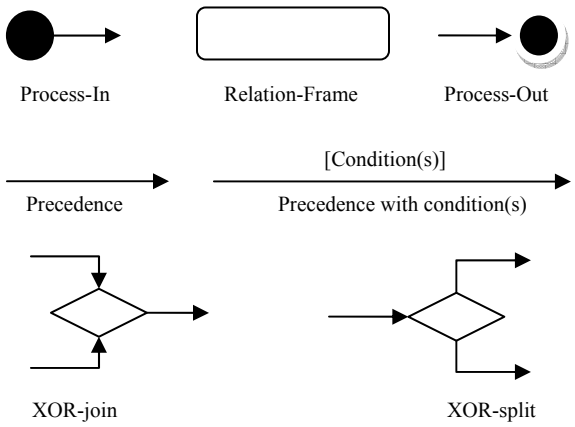

Figure 6: Some examples of notations for relations

In the Relation-Frame, the precedence and logical relationships that tie SMUs (see Fig. A1) are represented to show the flow-paths for the entities, including the conditions that create the alternative flow-paths. This relation-view provides the simulation developers the conceptual foresee of the entity flow in the sub-system, which supports the verification of logic associated with SMUs.

Finally, the sequence-diagram section shows a series of messages exchanged by a selected set of objects in SMUs, with an emphasis on the chronological course of communication between SMUs – which is used to indicate the status and the responding sequences from taking an action (operation) of the objects related to SMUs. Some crucial notations are shown in Figure 7.

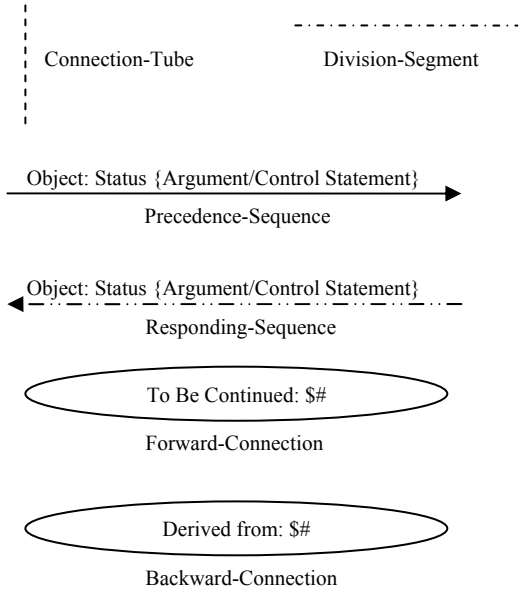

Figure 7: Notations for the sequence-diagram section

The sequence diagram, on the other hand, can be determined as a conceptual simulation that illustrates a brief simulation run. It has (see Fig. A1 for a better understanding) both Begin and End runs, initial set-up for variables, entities and their flows, resource utilization, variable changes, activities, and time-sequences (divided by division-segment and prioritized activity orders). This diagram is also used to pre-check whether or not individual or a set of SMUs have sufficient parameters (e.g., entity/resource type, variables, and operations) to fulfill the simulation requirements prior to further DSSE development.

The later step is to provide the descriptions of the objects used in these sections in a tabular form (table). Each table gives not only an object's generic information (e.g., name, type, description, and associated parameters) but also its extension (e.g., event state, rules, and algorithms) if needed. There is no specific regulation in designing a table of description. The design is depended on the demand and detailed level of information.
(Note: due to the size of the tables and figures, they are partially shown in the appendices section for DMSL:RINV as an example.)

The final step is to revise every section and connect them together by using Connection-Tube. This line con-

tains data given in each SMU and passes them throughout its length. Thus, the simulation developers are able to keep tracks of every action and transaction state of the objects, by following the lines (Top-to-Bottom or Bottom-to-Top relation) and other associated notations (Left-to-Right or Right-to-Left relation), which helps support their conceptual thinking. It is seen that the logic behind the development of the ISAP process layer is to access a domain from a very generic component to sub-components with different detailed levels and to maintain the completeness of encapsulation and inheritance of component data for the component reusability. This means that ISAP well deploys the decomposition approach to remove the complexity of conceptual thinking as well as the composition approach to extend the scope of conceptual thinking.

The results of the connection and association of these SMUs, notations, and descriptions are transformed into a network statement. Here is the network statement of DMSL: RINV, as shown below.

Ref# 0:

1 SetReorderPoint, Reorder point;
2 CreateSignal, Arrival rate, Time of first arrival, Max # of demands;
3 CheckInventory, Resource#, File resource#, Inventory position;
4 Condition, INV_POS <= REORDER_PT;
5 SetOrder, Order quantity;
6 UpdateInventory, Inventory position;
7 RouteOrder, Lead time;
8 RecordSafetyStock, File#, Resource#, File resource#, Number of radios;
9 UpdateRadioStock, Resource#, File resource#, Number of radios;
10 TerminateSignal, Max# of signals;
11 Condition, INV_POS > REORDER_PT;
12 TerminateSignal, Max# of signals;

Each line of the network statement contains sequential-order numbers and operation names with their parameters. Using a network statement is a basic idea found in the structure of commercial simulation software such as Arena© (Kelton, Sadowski, and Sadowski 2002) and AweSim© (Pritsker and O'Reilly 1999) to create simulation modeling frameworks for the construction and control of simulation modules. Therefore, a simulation modeling framework defined by the network statement and by other aspects through the ISAP development process can be used to generate appropriate simulation modules for the DSSE development.

4 CONCLUSIONS

The specialization of the CM concepts and techniques is taken as the main idea of this research study to improve the CSM approach. This is because CSM has been seen as a critical approach that is used to shorten gaps of communication between the domain experts and the simulation developers and to reduce difficulties of transformation of the concepts between two different domains of knowledge. However, CSM has been largely ignored, especially when conducted the development of DSSE. ISAP is a prototype that is developed based on the conceptual modeling approach under the SE and KE disciplines to support the development of a conceptual simulation model. Moreover, ISAP is designed to match with the structural and behavioral characteristics of the DSSE development process. Thus, simulation developers can apply ISAP to generate robust and reusable simulation modeling frameworks that can be used as blueprints giving designs and instructions for the specific simulation development projects. Nevertheless, there are still more rooms for improvement for this ISAP prototype to fulfill other simulation requirements, for example, dynamic parameter assignment, random distributed data generation, and simulation-module interface. For this study, the ISAP prototype is expected by the authors that it is able to encourage the simulation developers to enhance the current capability of the available modeling methods/tools to take simulation development to higher level.

ACKNOWLEDGMENTS

The authors would like to thank the Center for Systems Modeling and Simulation at Indiana State University, Kang-Hung Yang, and Zhilli Zhou (Computational Optimization and Logistics Lab, the University of Oklahoma) for their help and cooperation.

APPENDICES

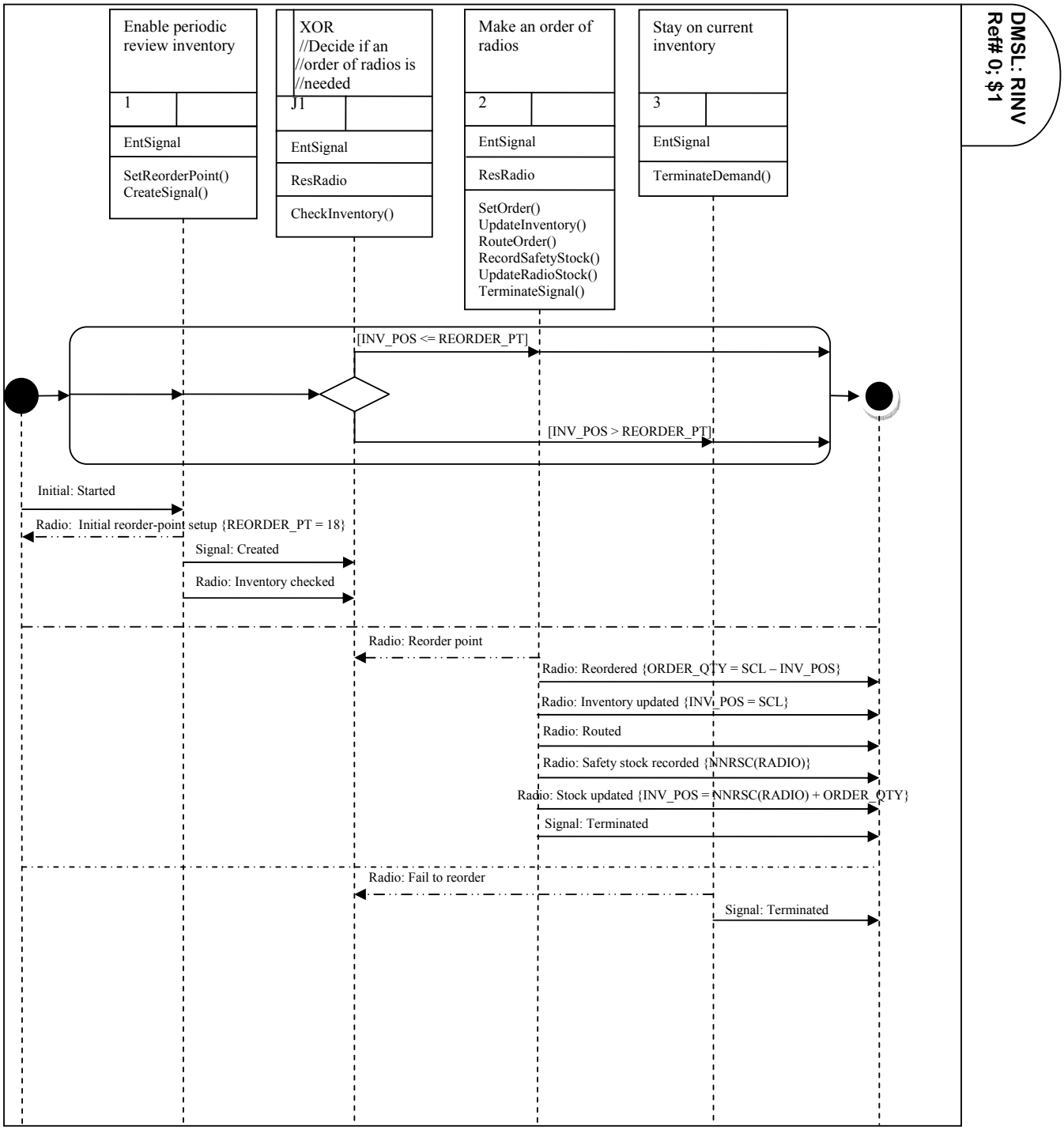

Figure A1: An example of DMSL: RINV

Table A1: Description of objects for DMSL: RINV.

Object Name	Type	Description	Parameters
EntSignal	Entity	This object represents a signal entity to enable the periodic review-reorder inventory system.	: Arrival rate
ResRadio	Resource	This object represents radio resources that can be altered corresponding to the inventory status.	: Resource# : Resource capacity : Queue# : Queue capacity

Table A2: Description of operations for DMSL: RINV.

Operation Name	Actor	Description	Attributes	Global Variables
CheckInventory()	ResRadio	An action is to determine if radio is available to satisfy a customer demand.	N/A	: Resource# :File resource# :Inventory position
CreateSignal()	EntSignal	A signal entity is created to the systems.	: Arrival rate : Time of first arrival : Max# of signal entities	
SetReorderPoint()	N/A	An action is to set a reorder point for the inventory system.	N/A	: Reorder point
SetOrder()	N/A	An action is to set a quantity of order	N/A	: Order quantity
RecordSafetyStock()	ResRadio	Number of radios are available at that time of arrival of shipment.	N/A	: File# : Resource# : File resource# : Number of radios
RouteOrder()	N/A	A quantity of order is transport to the discount house's inventory with a lead time	N/A	: Lead time
UpdateInventory()	N/A	Number of radios in the inventory are updated	N/A	: Inventory position
UpdateRadioStock()	ResRadio	Number of radios are updated.	N/A	: Resource# : File resource# : Resource capacity
TerminateSignal()	EntSignal	Each signal entity is terminated.	N/A	: Max# of signals

Table A3: Description of variables for DMSL: RINV.

Variable	Equivalence	Description
INV_POS	Inventory position	It contains the overall number of radios derived from both sub-systems.
NNRS(RADIO)	Number of radios	It shows the exact number of radios at the physical inventory.
ORDER_QTY	Order quantity	It indicates the number of radios per an order.
REORDER_PT	Reorder point	It sets the minimum number of radios in the physical inventory for reorder.
SCL	Stock control level	It limits the maximum number of radios in the physical inventory.

REFERENCES

Bartholet, R. G., D. C. Brogan, P. F. Jr. Reynolds, and J. C. Carnahan. 2004. In search of the philosopher's stone: Simulation composability versus component-based software design, In *Proceedings of the Fall 2004 Simulation Interoperability Workshop*, Orlando, Florida.

Bell, J., N. Snooke, and C. Price. 2005. Functional decomposition for interpretation of model based simulation. In *Proceedings of the 19th International Workshop of Qualitative Reasoning*, 192-198.

Benjamin, P. C., M. Blinn, F. Fillion, and R. J. Mayer. 1993. Intelligent support for simulation modeling: a description-driven approach. In *Proceedings of the 1993 Summer Computer Simulation Conference*.

Booch, G., I. Jacobson, and J. Rumbaugh. 1999. *The Unified Modeling Language user guide*. Massachusetts:Addison-Wesley.

Brooks, R. J. 2006. Some thoughts on conceptual modeling: Performance, complexity and simplification. In *Proceedings of the 2006 OR Society Simulation Workshop*.

Butler, B. 1998. Simulation composability for JSIMS, In *Proceedings of the 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications*, 4-14.

Coleman, D., P. Arnold, S. Bodorff, C. Dollin, and H. Gilchrist. 1993. *Object oriented development: The fusion method*. Prentice Hall.

Cyre, W. R. 1999. Conceptual Modeling and Simulation. In *Proceedings of the 1999 IEEE International Conference on Computer Design*, 293-296.

Davis, W. J. 2001. Distributed simulation and control: The foundations. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer. 187-198. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Deursen, A. V., P. Klint, and J. Visser. 2000. Domain-specific languages: An annotated bibliography. Available via <http://homepages.cwi.nl/~arie/papers/dslbib/> [accessed February 14, 2008].

Dieste, O., N. Juristo, A. M. Moreno, and J. Pazos. 2001. Conceptual modeling in software engineering and knowledge engineering: Concepts, Techniques and trends. *Handbook of Software Engineering & Knowledge Engineering* 1: 733-766. New Jersey: World Scientific.

Grant, F. H. and A. A. B. Pritsker. 1974. Models of cadmium electroplating processes. *NSF (RANN) GRANT GI-35106*, Purdue University.

Heavey, C. and J. Ryan. 2006. Process modeling support for the conceptual modeling phase of a simulation project. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 801-808. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hofmann, M. A. 2004. Criteria for decomposing systems into components in modeling and simulation: Lessons learned with military simulations. *Simulation* 80: 357-365.

Iba, T., Y. Matsuzawa, and N. Aoyama. 2004. From conceptual models to simulation models: Model Driven Development of Agent-Based simulations. In *Proceedings of the 9th Workshop on Economics and Heterogeneous Interacting Agents*. 1-12 Kyoto, Japan.

Kasputis, S. and H. C. Ng. 2000. Composable simulations. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick. 1577-1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Kelton, W. D., R. P. Sadowski, and D. A. Sadowski. 2002. *Simulation with Arena*. 2nd ed. New Yok: McGraw Hill.

Lee, J. and G. M. Wyner. 2003. Defining specialization for data flow diagrams. *Information Systems* 28: 651-671.

Meyer, B. 1997. *Object-oriented software construction*. 2nd ed. London: Prentice Hall.

Miner, R. J. and F. H. Grant. 1978. *User's guide for SNAP*. Pritsker & Associates, Inc.

Pace, D. K. 2000. Ideas about simulation conceptual model development. *Johns Hopkins APL Technical Digest* 21: 327-336.

Page, E. H. and J. M. Opper. 1999. Observations on the complexity of composable simulation. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans. 553-560. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Petty, M. D. and E. W. Weisel. 2003. A composability lexicon, In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando, Florida.

Pritsker, A. A. B. and J. J. O'Reilly. 1999. *Simulation with Visual SLAM and AweSim*. 2nd ed. New York: John Wiley & Sons.

Robinson, S. 2006a. Conceptual modeling for simulation: Issues and research requirements. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 792-800. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Robinson, S. 2006b. Issues in conceptual modeling for simulation: Setting a research agenda. In *Proceedings of the 2006 OR Society Simulation Workshop*.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. 1991. *Object-oriented modeling and design*. New Jersey: Prentice Hall.

Saanen, Y. A. 2004. *An approach for designing robotized marine container terminals*, Doctoral Dissertation, Delft University of Technology, Netherlands.

Sarjoughian, H. and D. Huang. 2005. A multi-formalism modeling composability framework: Agent and discrete-event models. In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real Time Applications*. 249-256.

Setavoraphan, K. 2005. Conceptual simulation modeling for regional distribution center systems, Thesis, Indiana State University, Indiana, USA.

Setavoraphan, K. and F. H. Grant. 2008. *Simulation model for lockage operation*, contributed to the INFORMS Southwest Regional Conference, Texas A&M University, Texas, USA.

Spiegel, M., P. F. Reynolds, and D. C. Brogan. 2005. A case study of model context for simulation composability and reusability, In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Teeuw, W. B. and H. van den Berg. 1997. *On the quality of conceptual models*. Available on < `http://osm7.cs.byu.edu/ER97/workshop4 /tvdb.html>` [accessed February 14, 2008].

Valentin, E. C. and A. Verbraeck. 2002. Guidelines for designing simulation building blocks. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Valentin, E. C. and A. Verbraeck. 2005. Requirements for domain specific discrete event simulation environments. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Verbraeck, A. and A. Dahanayake. 2002. *Building blocks for Effective Telematics Appliation Development and Evaluation* (BETADE), Delft University of Technology, Netherlands.

Verbraeck, A. and E. Valentin. 2002. Simulation building blocks for airport terminal modeling. In *Proceedings of the 2002 Winter Simulation Conference*, ed. E. Yucesan, C. –H. Chen, J. L. Snowdon, and J. M. Charnes. 563-571. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Vreede, G. J., A. Verbraeck, and D. T. T. V. Eikck. 2003. Integrating the conceptualization and simulation of business processes: A modeling method and arena template. *SIMULATION* 79: 43-55.

Wang, W. and R. J. Brooks. 2006. Improving the understanding of conceptual modeling. In *Proceedings of the 2006 OR Society Simulation Workshop*.

Winnell, A. and J. Ladbrook. 2003. Towards composable simulation: Supporting the design of engine assembly lines. In Proceedings of the 17th European Simulation Multiconference, 431-436.

Yilmaz, L. and T. I. Oren. 2004. A conceptual model for reusable simulations within a model –simulator-context framework. In *Proceedings of the Conference on Conceptual Modeling and Simulation*, Part of the 13M – International Mediterranean Modeling Multiconference. 235-241.

Zhou, M., Y. J. Son, and Z. Chen. 2004. Knowledge representation for conceptual simulation modeling. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters. 450-458. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Zhou, M., K. Setavoraphan, and Z. Chen. 2005. Conceptual simulation modeling of warehousing operations. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines. 1621-1626. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Zhou. M., Q. Zhang, and Z. Chen. 2006. What can be done to automate conceptual simulation modeling?. In *Proceedings of the 2006 Winter Simulation Conference*, ed. L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. 809-814. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

KITTI SETAVORAPHAN is a Ph.D. candidate in the School of Industrial Engineering at the University of Oklahoma. Kitti received his BS degree in Computer Engineering from Assumption University, Thailand, in 2002, and MS degree in Industrial Technology from Indiana State University in 2005. He was a research assistant at the Center for System Modeling and Simulation, Indiana State University, focusing on modeling distribution operations. Currently, he is working at the Center for Wireless EMC, the University of Oklahoma, focusing on the study of the electromagnetic interactions of wireless communication devices. His research interests include simulation language/environment development and business process modeling/engineering for inland waterway transportation and intermodal container terminal systems. His email address is `Kitti.Setavoraphan-1@ou.edu`.

FLOYD H. GRANT is the Dugan Professor of Industrial Engineering at the University of Oklahoma. His research interests include simulation language development and Supply Chain Optimization. He is also the Director of the Center for Wireless EMC, focusing on finding solutions to cell phone interactions with other electronics. He is also founder of FACTROL, a company focused on managing the factory floor using simulation technology. Dr. Grant can be connected at `hgrant@ou.edu`.