Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds.

A SIMPLE EXAMPLE OF SYSML-DRIVEN SIMULATION

Leon McGinnis

Sch. of Industrial & Systems Engineering
Georgia Institute of Technology
Atlanta, GA 3032-0205, USA

Volkan Ustun

Sch. of Industrial & Systems Engineering
Georgia Institute of Technology
Atlanta, GA 3032-0205, USA

ABSTRACT

The successful practice of simulation requires a number of capabilities; two key capabilities are creating a conceptual model of the system to be simulated, and translating the conceptual model to a computational process or simulation program. We describe how OMG's new graphical systems modeling language, OMG SysML™ (OMG 2009), can be used to create a conceptual model, and how this conceptual model can be translated automatically to a simulation program. In demonstrating the process, we use Arena™ as the target simulation language, but the concepts presented are quite general.

1 INTRODUCTION

The practice of simulation requires a set of technical skills, capabilities, and processes. A number of authors have described simulation practice in terms of a set of steps or processes that must be executed. One example of such a description is summarized in Table 1, which is based on (Nubile, Ambrose and Mackarel 2004).

Table 1: Modeling and simulation workflow

1. **Problem Definition.** Clearly defining the goals of the study so that we know the purpose,
2. **Project Planning.** Being sure that we have sufficient and appropriate resources to do the job.
3. **System Definition.** Determining the boundaries and restrictions
4. **Conceptual Model Formulation.** Developing a preliminary model either graphically or in pseudo-code to define the components, and interactions (logic) that constitute the system.
5. **Preliminary Experimental Design.** Selecting the factors to be varied, and the
6. **Input Data preparation.** Identifying and collecting the input data
7. **Model Translation.** Formulating the model in an appropriate simulation language
8. **Verification.** Does the simulation correctly represent the data inputs and outputs?
9. **Validation.** Can the model be substituted for the real system for the purposes of experimentation?
10. **Final Experimental Design**. Designing an experiment that will yield the desired information
11. **Experimentation**. Executing the simulation to generate the desired data and sensitivity analysis.
12. **Analysis and Interpretation**. Drawing inferences from the data generated by the simulation runs.
13. **Implementation and Documentation**. Reporting the results, putting the results to use,

Clearly, the practice of simulation is not as simple as the linear execution of these thirteen steps—there usually is considerable iteration. Our focus is on steps 4 and 7, formulating the conceptual model which describes the relevant aspects of the system to be simulated, and translating the conceptual model to a suitable simulation language for computation. Traditionally, both these steps have been taught and viewed as the "art" or "craft" of simulation, i.e., as skills that are difficult to teach because there are few systematic tools to support them.

There is good reason to believe this situation could change. In the software engineering community, model driven architecture, or MDA, is transforming the way software systems are designed and implemented. Wikipedia describes MDA as providing "a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems." In other words, the MDA ap-

proach brings a significant level of formalism to the process of developing the conceptual model of the intended software system, as well as formal tools that automate a significant portion of the implementation coding.

Can the same results be achieved by applying the MDA approach to the development of simulations? In this paper, we demonstrate that a tentative answer is "yes" by showing how SysML can be used to develop a domain specific language for creating conceptual models in a particular domain, and how data interchange standards like XML/XMI can be used to support the automatic translation of a model in the domain specific language to a model in a simulation language..

2 OVERVIEW

A "domain specific language," or DSL, is described by Wikipedia as "a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique." The domain could be an abstract domain, such a queuing theory, or a real domain, such as semiconductor manufacturing. It even could be a programming domain, such as discrete event simulation. In fact, simulation languages, like Arena™, are domain specific languages.

Figure 1 illustrates how domain specific languages might be used to enhance steps 4 and 7 in Table 1. Two distinct domain specific languages are represented in Figure 1. On the left is a DSL dedicated to the description or specification of semiconductor manufacturing scenarios, which is used to create a specific instance model or specification for a particular semiconductor manufacturing situation. On the right is a discrete event simulation DSL, or simulation language, which can be used to create specific simulation models or simulation programs. In the middle are two ovals. The top oval represents a mapping between the two domain specific languages, e.g., a specification of the relationships between a syntactical construct in the semiconductor manufacturing DSL and an equivalent syntactical construct in the discrete event simulation DSL. The bottom oval in Figure 1 represents a computational process for converting a specific instance model of a semiconductor manufacturing model into a specific instance of a discrete event simulation model.

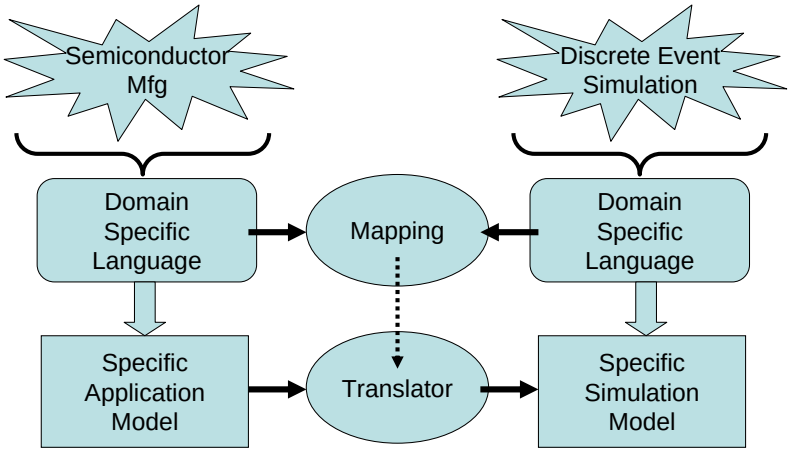

Figure 1: Framework for Model-Driven Simulation

There are a number of challenges in implementing the framework illustrated in Figure 1. Perhaps the most obvious is creating a DSL for the application domain. The fact that almost every discussion of the process of "doing simulation" (as described in Table 1) provides relatively vague description of step 4 is some evidence for the lack of explicit domain specific languages in general. SysML has been suggested as a basis for a semiconductor manufacturing DSL in (Huang *et al* 2008); although they did not use the term "domain specific language" their description is entirely consistent with the intent of the term.

SysML has a number of advantages to recommend it as a basis for an application specific DSL. It is a graphical language, which makes it possible to discuss SysML models with the "problem owners" in order to validate modeling assumptions. Also, SysML is a formal language—it has a formal syntax, which enables a number of computational tools which operate on SysML models. SysML is implemented as a profile of UML, and as a result there are many commercially available tools for SysML. SysML is easily extended and specialized, an important advantage in developing a DSL. SysML has an open, or publically available standard, making it easy to develop application which use SysML models.

Suppose an application-specific DSL has been developed, e.g., in SysML. A second major challenge is creating the mapping between this DSL and a discrete event simulation language. Part of the challenge is that most simulation languages do not have a publically available formal specification, thus creating a formal mapping from an application specific DSL is an arduous task.

The final challenge, of course, is the development of the model translator, which uses the mapping to translate an application specific model written in the application specific DSL to a corresponding simulation model written in the specific discrete event simulation language.

3 THE SIMPLE EXAMPLE

We illustrate, in a very limited way, an implementation of the framework illustrated in Figure 1. We use SysML to create a DSL for a subset of queuing theory. This domain specific language is used to create specific queuing models. We use Arena™ as the target discrete event simulation language. Since Arena has an existing Access™ interface (i.e., Arena models can be exported to MS Access, and imported from MS Access, using a particular data schema), we use this interface, rather than attempting to create a "native" Arena DSL in SysML. We do, however, create an Access DSL in SysML, and use it to specify the schema required for Arena export/import.

Because we have chosen a relatively limited domain, the mapping from the SysML DSL to the Arena DSL is relatively straightforward. We address this issue in our concluding remarks.

Both SysML and MS Access have facilities for XMI file export and import, which simplifies the process of creating a translator. Our translator, which operates on the appropriate XMI files, is implemented in Atlas Transformation Language (ATL). Figure 2 summarizes the structure of the example implementation.

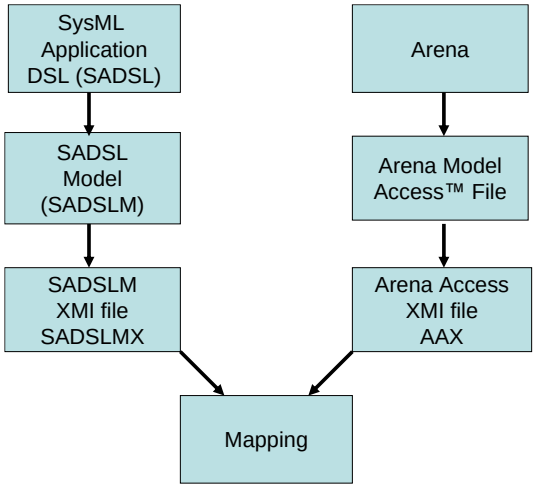

Figure 2: Structure of the Example

For this simple example, we have used the standard flow shop model as described in Huang et al. (2007). There are two workstations in sequence and each workstation consists of three identical machines in parallel. Topcased-SysML editor as a Eclipse plugin is used to develop the SysML models. SysML models are exported as XMI files using the Topcased SysML metamodel based on the Eclipse Modeling Framework's Ecore metametamodel. For demonstration purposes, we have created a simpler metamodel for the flow shop model using Ecore as the underlying metametamodel. This metamodel -named "SysMLCreate"- is depicted in Figure 3. The original XMI file is then transformed to a new XMI file using the "SysMLCreate" as the base metamodel. The SysML model captured in the transformed XMI file is shown in Figure 4.

Transformation of the SysML model to an MSAccess database representing Arena™ models requires creating a metamodel of the underlying database structure. The transformation is defined using Atlas Transformation Language (ATL). A simple rule written in ATL to create "Process" modules for Arena™ is presented in Figure 5. As a result of the transformation, an XMI file is created, which is then imported to MS Access. Importing the MS Access database to the Arena™ modeling environment completes the transformation. The resulting Arena™ model is depicted in Figure 6.

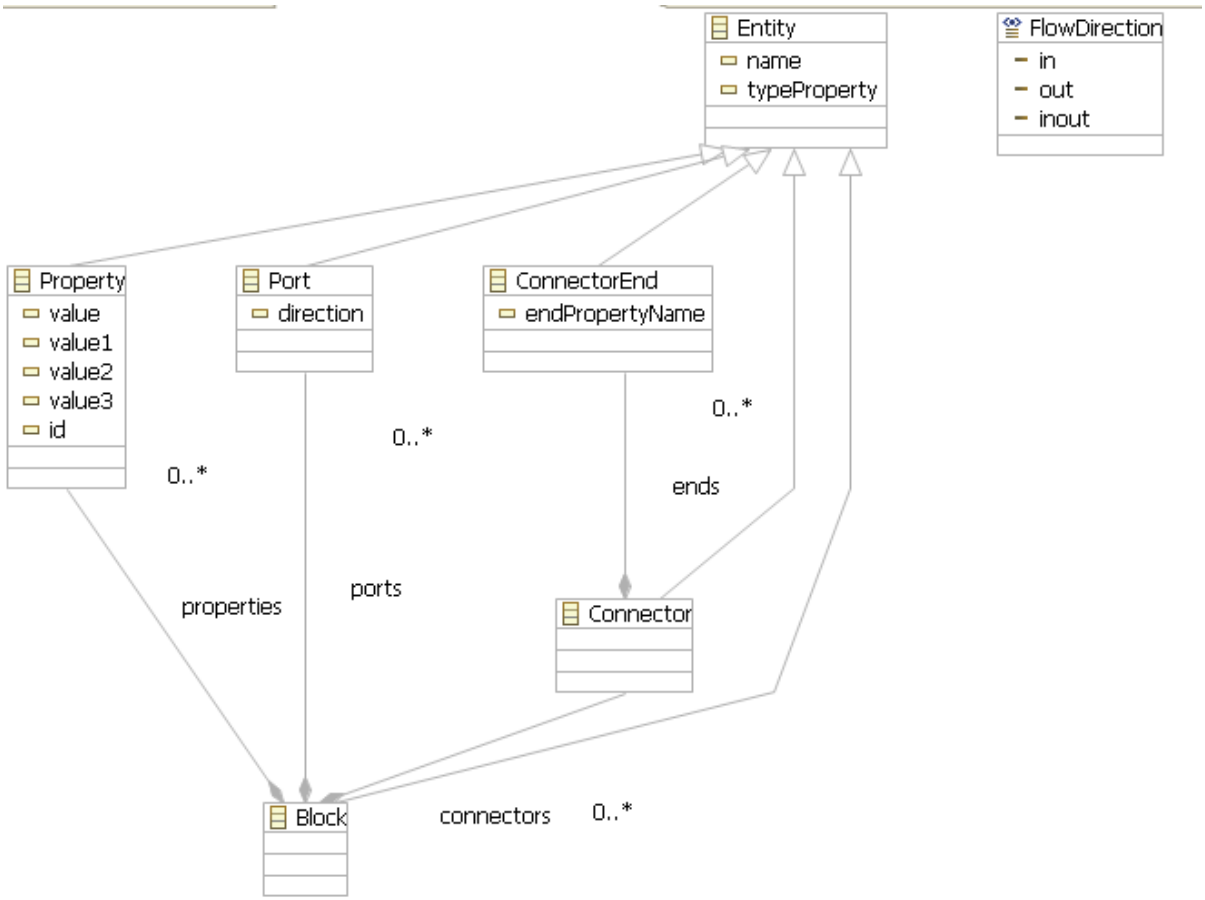

Figure 3: Simplified SysML metamodel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="SysMLCreated">

<Block name="Buffer"/>
<Block name="ArrivalProcess"/>
<Block name="Part"/>
<Block name="Machine"/>
<Block name="DepartureProcess"/>
<Block name="Workstation">
        <properties name="buffer1" typeProperty="Buffer"/>
        <properties name="machine1" typeProperty="Machine"/>
        <properties name="machine2" typeProperty="Machine"/>
        <properties name="machine3" typeProperty="Machine"/>
</Block>
<Block name="FlowShopSystem">
        <properties name="arrivalProcess1" value="Expo(5)" typeProperty="ArrivalProcess"/>
        <properties name="workstation1" id="1" value="Triangular" value1="5" value2="8" value3="10" typeProperty="Workstation"/>
        <properties name="workstation2" id="2" value="Triangular" value1="5" value2="8" value3="10" typeProperty="Workstation"/>
        <properties name="departureProcess1" typeProperty="DepartureProcess"/>
        <connectors name="connector1" typeProperty="Connector">
                        <ends endPropertyName="arrivalProcess1"/>
                        <ends endPropertyName="workstation1"/>
        </connectors>
        <connectors name="connector2" typeProperty="Connector">
                        <ends endPropertyName="workstation1"/>
                        <ends endPropertyName="workstation2"/>
        </connectors>
        <connectors name="connector3" typeProperty="Connector">
                        <ends endPropertyName="workstation2"/>
                        <ends endPropertyName="departureProcess1"/>
        </connectors>
</Block>

<Block name="Workstation">
        <properties name="machine1" typeProperty="Machine"/>
        <properties name="machine2" typeProperty="Machine"/>
        <properties name="machine3" typeProperty="Machine"/>
        <properties name="buffer1" typeProperty="Buffer"/>
        <ports name="PartIn" direction="in" typeProperty="Port"/>
        <ports name="PartOut" direction="out" typeProperty="Port"/>
        <connectors name="connector1" typeProperty="Connector">
                        <ends endPropertyName="PartIn"/>
                        <ends endPropertyName="buffer1"/>
        </connectors>
        <connectors name="connector2" typeProperty="Connector">
                        <ends endPropertyName="buffer1"/>
                        <ends endPropertyName="machine1"/>
        </connectors>
        <connectors name="connector3" typeProperty="Connector">
                        <ends endPropertyName="buffer1"/>
                        <ends endPropertyName="machine2"/>
        </connectors>
        <connectors name="connector4" typeProperty="Connector">
                        <ends endPropertyName="buffer1"/>
                        <ends endPropertyName="machine3"/>
        </connectors>
        <connectors name="connector5" typeProperty="Connector">
                        <ends endPropertyName="machine1"/>
                        <ends endPropertyName="PartOut"/>
        </connectors>
        <connectors name="connector6" typeProperty="Connector">
                        <ends endPropertyName="machine2"/>
                        <ends endPropertyName="PartOut"/>
        </connectors>
        <connectors name="connector7" typeProperty="Connector">
                        <ends endPropertyName="machine3"/>
                        <ends endPropertyName="PartOut"/>
        </connectors>
 </Block>


</xmi:XMI>
```

Figure 4: Transformed SysML model

```
module SysMLCreated2Arena; -- Module Template
create OUT : Queue from IN : SysMLCreated;


rule SysML22Process {
    from
        dt : SysMLCreated!Property (
            if dt.typeProperty='Workstation' then true
            else false
            endif
            )
    to
        out : Queue!BasicProcess_x007C_Process (
            Name <- dt.name,
            SerialNumber <- dt.id,
            ModelLevel <- '1',
            X<-'0',
            Y<-'0',
            ReportStatistics<-'Yes',
            Type<-'Standard',
            Action<-'SDR',
            ValueAdded<-'VA',
            DelayType <- dt.value,
            Units <-'Hours',
            Priority<-'2',
            Expression <-'1',
            StDev<-'.2',
            Max<-dt.value3,
            Min<-dt.value2,
            Value<-dt.value1
        )


}
```

Figure 5: SysML to "Process" module transformation

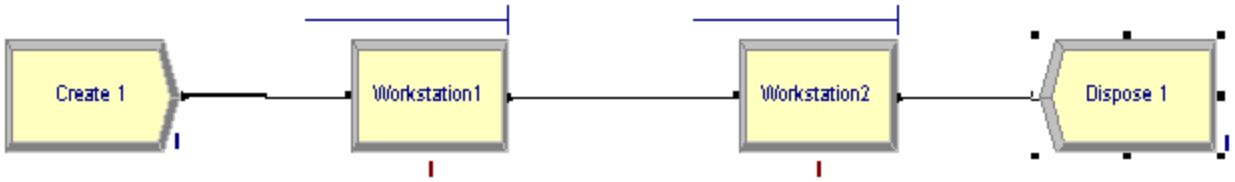

Figure 6: Generated Arena™ model

4 ISSUES

We have chosen an admittedly limited demonstration domain, so it is a fair question to ask whether or not this approach scales up. We believe the answer is a qualified "yes," for several reasons. First, based on the previous and ongoing research at Georgia Tech (Huang *et al* 2007, 2008) and elsewhere (see, e.g., Schönherr and Rose 2009), we believe there is little reason to doubt the feasibility of creating domain specific languages for usefully large application domains. In fact, the publication of the Core Manufacturing Simulation Data standard by NIST (Johnson *et al* 2007, Lee *et al* 2007, and Riddick and Lee 2008) is compelling evidence that the *structure* of manufacturing can be described in a formal way. Of course, the challenge of creating the formal syntax and semantics for behavior has yet to be definitively settled.

Second, while model translation in the practice of simulation traditionally has been an *ad hoc* process, there are developments in computer science that hold great promise for making the process much more formal. For example, research on graph transformation and triple graph grammars (see. e.g., Amelunxen *et al* 2008) has led to the creation of tools (see, e.g., <http://www.moflon.org/>) for creating mappings between metamodels and using the mappings to automate model translation.

While we believe the technology is emerging to enable the development of domain specific languages in manufacturing, and their automatic translation to simulation code, that does not mean there are not remaining issues. Here we highlight two specific issues:

(1) Some systems are very large, with many instances of the same entity or component. The creation and management of such large instance models represents a challenge of its own.

(2) In order to automatically translate a conceptual model (e.g., realized using a formal DSL) into its corresponding simulation program, all the information needed to create that simulation model must be contained in the conceptual model, because there is no human simulation analyst to elaborate terse descriptions, or disambiguate incomplete descriptions. Thus, the "programming" aspects of simulation will move in two directions—off line development of the DSL, the mapping, and the translator, and on-line application of the DSL to capture a fully rendered conceptual model.

5 CONCLUSION

Our purpose in this paper has been to demonstrate with a simple example how a formal modeling language, SysML, can be used to transform the practice of simulation by making the conceptual modeling process much more formal, and largely automating the creation of the simulation code. The reader may ask, "Why would you want to do this?" The motivation is simple—we seek to make the practice of simulation much more widespread than it is today. One strategy is to make simulation technology more accessible to the customer, by giving customers and simulation practitioners the tools they need to create complete descriptions of the problem to be simulated, in a form that can be understood by the customer, i.e., in the customer's language. If at the same time, the time, cost, and potential for error in "programming" simulation codes can be largely eliminated, then more time and attention can be focused on working with the customers to insure the problem is properly specified.

REFERENCES

Amelunxen, C., F. Klar, A. Königs, T. Rötschke, and A. Schürr. 2008. Metamodel-based tool integration with moflon. *Proceedings of the 30th international conference on Software engineering*. Leipzig, Germany. ACM.

Atlas Transformation Language (ATL). 2009. http://www.eclipse.org/m2m/atl/

Ecore. 2009. http://www.eclipse.org/modeling/emf/ [accessed July 29, 2009].

Haddock, Jorge. 1988. A Simulation Generator for Flexible Manufacturing Systems Design and Control. *IIE Transactions*. 20(1):22-31.

Huang, E., R. Ramamurthy, and L. McGinnis. 2007., System and Simulation Modeling Using SysML. In *Proceedings of the 2007 Winter Simulation Conference*. eds. S. G. Henderson, B. Biller, M.-H Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 796-803. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Huang, Edward, Ky Sang Kwon, and L. F. McGinnis, "Toward On-Demand Wafer Fab Simulation using Formal Structure and Behavior Models," *Proceedings of the 2008 Winter Simulation Conference*; ed S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler, 2341-2349. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Johansson, M., S. Leong, Y. T. Lee, F. Riddick, G. Shao, B. Johansson, A. Skoogh, and P. Klingstam. 2007. A Test Implementation of the Core Manufacturing Simulation Data Specification, In *Proceedings of the 2007 Winter Simulation Conference*, 1673-1681. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Lee, Y. T., S. Leong, F. Riddick, M. Johansson, and B. Johansson. 2007. A Pilot Implementation of the Core Manufacturing Simulation Data Information Model. In *Proceedings of the Simulation Interoperability Standards Organization 2007 Fall Simulation Interoperability Workshop*. Orlando, Florida: Simulation Interoperability Standards Organization, Inc.

Nubile, E., E. Ambrose, and A. Mackarel. 2004. Development of a procedure for manufacturing modeling and simulation. 21st International Manufacturing Conference, Limerick, Ireland.

Object Management Group. 2009. OMG SysML™ v1.1. Avaliable via http://www.omg.org/spec/SysML/1.1/ [accessed July 29, 2009].

Riddick, F. and Y. Lee. 2008. Representing layout information in the CMSD specification. *Proceedings of the 2008 Winter Simulation Conference*, ed S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler, 1777-1784. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

TOPCASED. 2009a. Available via www.topcased.org [accessed July 29, 2009].

TOPCASED. 2009b. SysML metamodel implementation. Available via www.topcased.org [accessed July 29, 2009].

Wikipedia. 2009a. Model-driven Architecture. Anon. http://en.wikipedia.org/wiki/Model_Driven_Architecture [accessed July 29, 2009].

Wikipedia. 2009b. Domain-specific Language. Anon. http://en.wikipedia.org/wiki/Domain_specific_language [accessed July 29, 2009].

AUTHOR BIOGRAPHIES

LEON MCGINNIS is the Gwaltney Professor of Manufacturing Systems at the Georgia Institute of Technology, where he serves as Associate Director of the Manufacturing Research Center, Director of the Product and Systems Lifecycle Management Center, and founder of the Keck Virtual Factory Lab. His personal research focuses on systems modeling and systems design for discrete event logistics systems, and he leads several research teams addressing large scale systems analysis problems for industry sponsors. He is a member of IEEE, INFORMS, and IIE. His email address is <leon.mcginnis@gatech.edu>.

VOLKAN USTUN is a postdoctoral research fellow in the H. Milton Stewart School of Industrial and Systems Engineering at the Georgia Institute of Technology. He has received his B.S. and M.S. degrees in Industrial Engineering from Middle East Technical University (METU), Turkey and his Ph.D. degree in Industrial and Systems Engineering from Auburn University in 2009. Prior to joining the Ph.D. program, he has worked as a software engineer at The Scientific and Technical Research Council of Turkey (TUBITAK). His research interests mainly include discrete-event and agent-based simulation models and frameworks for complex systems. His email address is <volkan.ustun@isye.gatech.edu>.