Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

SYSTEM AND SIMULATION MODELING USING SYSML

Edward Huang
Randeep Ramamurthy
Leon F. McGinnis

School of Industrial & Systems Engineering
765 Ferst Drive, N.W.
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

ABSTRACT

Simulation languages and the GUIs supporting them may be excellent tools for creating simulation codes, but are not necessarily the best tools to use for creating descriptions of systems, i.e., for modeling. In 2006, OMG published the initial standard specification (OMG 2006) for SysML (Systems Modeling Language), an extension of UML (OMG 2007) designed specifically to support systems engineering. SysML shows great promise for creating object-oriented models of systems that incorporate not only software, but also people, material, and other physical resources, expressing both structure and behavior for such systems. In this paper, we explore the use of SysML both to model a system to be simulated and to support the automatic generation of simulation models.

1 INTRODUCTION

A fundamental problem in simulating complex systems is verifying that the simulation model adequately describes the system being simulated (Law and Kelton 2000). At least part of the difficulty is that while the simulation model is based on a formal language, there is no corresponding formal description of the system being simulated. Thus, the process of verification becomes largely *ad hoc*.

In 2006, OMG published the initial standard for SysML (Systems Modeling Language), an extension of UML designed to support systems engineering in general, but specifically system modeling. SysML is a graphical language for building models of large-scale, complex, and multi-disciplinary systems, and there already are a number of commercial tools supporting SysML.

SysML is appealing for a number of reasons. First, it is object-oriented, thus using it encourages and supports an object-oriented view of systems. Because it is based on UML, there are (or will be) many SysML resources available to students and potential users, including highly evolved software tools such as Embedded Plus Engineering's SysML toolkit (EmbeddedPlus© 2006) for the Rational Software Delivery Platform (IBM 2004). Also, because it is based on UML, there is the intriguing notion of "executable" models, or using the SysML model to automate simulation software generation.

We note that software generation from UML models is a common practice. However, because SysML is a new (and still evolving) extension of UML, there are, at the time of this writing, no examples we can find of discrete event simulation model generation from a SysML model.

Because SysML is so new, there is scarcely any experience with using it to model the kinds of systems that are often the target for simulation. Likewise, there are not yet any SysML textbooks, and the only examples to be found are those contained in the standard specification document, or in presentations by the OMG. The goal of this paper is to illustrate the potential for SysML as a modeling tool to support discrete event simulation.

We explore the use of SysML as a tool for creating formal models of systems to be simulated, and for automating the creation of the corresponding simulation models. The reader may well ask "Why introduce another language if you ultimately intend to create a simulation model in a specific target simulation language?" There are two responses. First, if the SysML model is easier to understand, in terms of its description of the system being simulated, then using SysML addresses the common problem of model verification, since the system model is translated "automatically" into its corresponding simulation model. Second, a common system model might provide a common basis for other analyses or system design.

The paper is organized as follows. Section 2 presents a very brief overview of SysML. Section 3 introduces a very simple example problem, a flowshop. Section 4 presents the essential SysML model of the flowshop, and Section 5 describes how the SysML model is translated

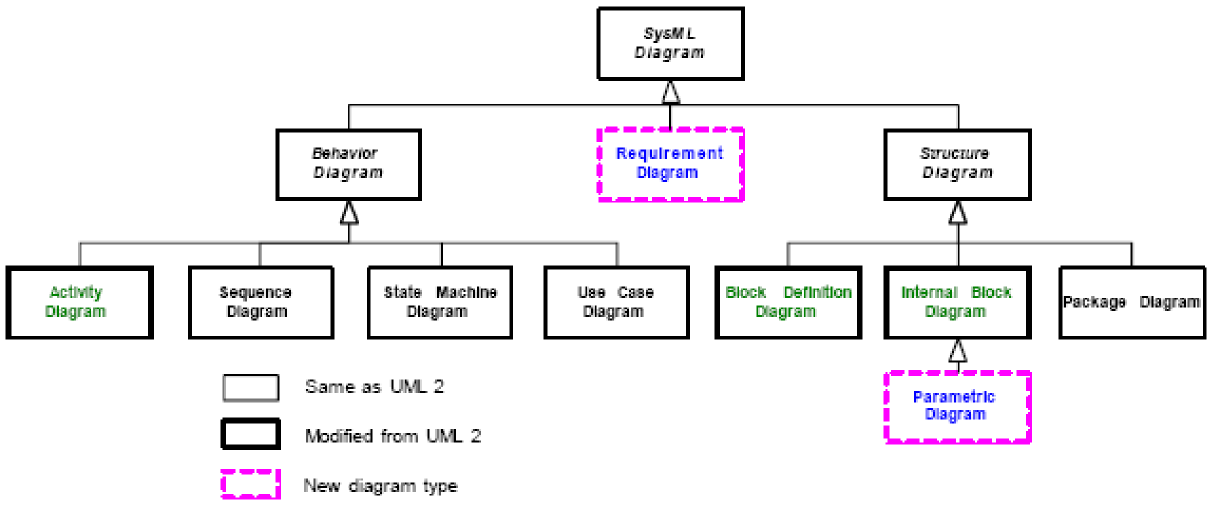


Figure 1: SysML diagram taxonomy (from OMG 2006).

into a corresponding eM-Plant simulation code. The paper concludes with a discussion of the potential for using SysML to support simulation.

2 SYSML DIAGRAMS

SysML re-uses a subset of UML, and adds some new diagrams specifically designed to support systems engineering. The basic organization of SysML diagrams is summarized in Figure 1, which is taken from (OMG 2006).

As shown in Figure 1, there are three groups of diagrams: structure, behavior, and requirements. In this initial study, we focus on structure.

According to the language specification, "blocks" in SysML: "are modular units of a system description, which define a collection of features to describe a system or other elements of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit" (OMG 2006).

Three diagram types express structure. The Block Definition Diagram is like an entity-relationship diagram, showing, e.g., how one block is composed from or related to other blocks. The Internal Block Diagram gives greater detail regarding the specific nature of the relationships between blocks, e.g., showing the details of flows or information transfer among blocks. The Parametric Diagram is used to capture the equations involved in the calculation of one or more parameters/attributes of the block.

Behavior is captured in three other diagram types. Sequence diagrams and state machine diagrams are taken directly from UML. The activity diagram is a modification of the UML activity diagram.

3 EXAMPLE: STANDARD FLOW SHOP

As a simple vehicle for demonstrating SysML and simulation model generation, we will use the standard flowshop model. A flowshop consists of a series of machine groups. Jobs flow in sequence from one machine group to the next. At any machine group, there may be a number of identical machines in parallel, and the next job to be processed claims the first available machine. For the jobs waiting for service at a machine group, the order of service is determined by a dispatching rule, such as first-come-first-served (FCFS).

4 SYSML MODELING

We develop two types of models. We create a *domain meta-model* of the flowshop application illustrated in Figure 2. The domain meta-model is developed independently of the specific analysis to be performed, and the goal is for one domain meta-model to linked to multiple analyses, e.g., both simulation and queuing models. We also create an *analysis meta-model* for the analysis to be performed. For example, if we use simulation, the analysis meta-model should describe the essential elements required to simulate a system. Given the domain meta-model and the analysis meta-model, we then map these meta-models to one another. If an instance of a system is described using the elements from the domain meta-model, then the corresponding analysis elements are also mapped to the instance model.

4.1 Domain Meta-model

From Figure 2, the different blocks of the system can be identified – arrival process, buffer, machine, workstation and the flowshop system itself. All these blocks can be captured using the Block Definition Diagram (BDD). Each block can have a list of parts, references or value types. In our example, the workstation block will have a buffer and one or more machines. Figure 3 shows the BDD of the flowshop system.

The detailed relationships between the different parts in a block can be captured using an Internal Block Diagram (IBD). In the workstation block of the flowshop system example, parts will enter the buffer first and then can move into any one of the 3 machines. Figure 4 and Figure 5 capture the internal relationship of parts within the workstation block and flowshop system block respectively.

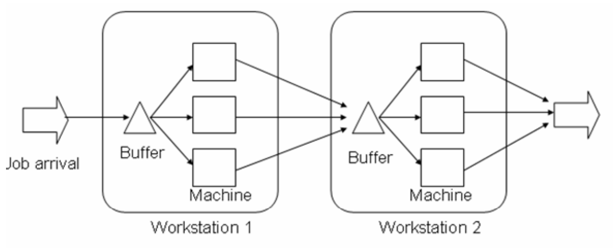


Figure 2: The flow shop system sketch.

4.2 Analysis Meta-models

Figure 6 shows the BDD for a simulation meta-model. A simulation model must consist of arrival process, buffers, tools, and jobs. This meta-model is independent of the simulation package that we wish to use.

Queuing analysis would require a set of constraint blocks (parametric equations) representing calculations for utilization, cycle time and WIP. A possible meta-model for this analysis is shown in Figure 7

4.3 Mapping the Domain Model and the Analysis Meta-model

We map the domain model to the analysis meta-model using the relationship semantics provided by SysML. These can be generalization, composition association, and aggregation association. In the flowshop example, the part block will be a generalization of the SimulationEntity block. This means that the part block will inherit all the properties and characteristics of the simulation entity block. The mapping of the flowshop domain model with the simulation and queuing analysis meta-model are shown in Figure 8 and Figure 9 respectively.

5 SIMULATION MODEL GENERATION

When the domain model is mapped to the simulation meta-model we can generate the actual simulation model automatically. We use an XMI export of the combined SysML model. XMI is an OMG standard for describing the SysML models.

We use Xpath (W3C 1999) to process the XMI file and generate a database with necessary information to be input to the simulation package. Finally, based on the simulation meta-model, a script is created in the target simulation language which allows the corresponding simulation package to read the data from the database and using the script, create the instance simulation program. Figure 10 specifies the architecture.

The parser application will check each block in the XMI file for the type of analysis used. For example, if the simulation meta-model mapping is found, the corresponding simulation object model will be generated. The logic is shown in Figure 11.

Thus the flowshop SysML model is automatically converted into a simulation model. Figure 12 shows the automatically generated flowshop model in the eM-Plant (UGS^{TM} 2003) simulation package.

6 CONCLUSION

System modeling and simulation modeling traditionally have been performed using quite different tools, e.g., relatively informal graphical diagrams for system description, and formal languages for simulation. The emergence of SysML may provide a way to create formal system models, and link them directly to formal simulation languages. The potential for this approach is quite interesting, as it would permit users to develop system models that could be automatically "parsed" into specific analysis models.

In this paper, we have presented one way in which this concept might be realized, and illustrated it with a simple flowshop example. Clearly, there are many issues, some fundamental and some implementation, to be addressed before this concept becomes a common practice. A fundamental issue is the representation of control mechanisms in the domain meta-model, and the "meta-mapping" of the control representation to the analysis models. This is an area of active research. In terms of implementation, one urgent issue is the development of standard meta-models for particular domains. These standard meta-models would provide the mechanism for integrating the vast research literature related t simulation applications. This also is an area of active research.

The integration of SysML and discrete event simulation models for manufacturing and logistics is very promising. The potential for redirecting the focus from programming in a particular simulation language to articulating a system model in a formal, but analysis-independent language provides a way to put simulation "on the desktop" of decision makers in a way that is not possible with the traditional craft approach to simulation modeling.

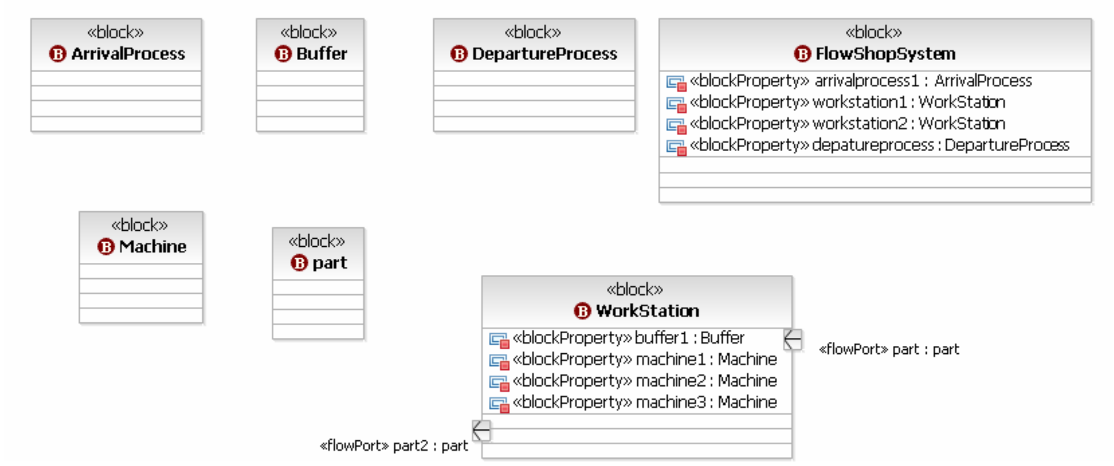

«block» ArrivalProcess
«block» Buffer
«block» DepartureProcess
«block» FlowShopSystem
«blockProperty» arrivalprocess1 : ArrivalProcess
«blockProperty» workstation1 : WorkStation
«blockProperty» workstation2 : WorkStation
«blockProperty» depatureprocess : DepartureProcess
«block» Machine
«block» part
«block» WorkStation
«blockProperty» buffer1 : Buffer
«blockProperty» machine1 : Machine
«blockProperty» machine2 : Machine
«blockProperty» machine3 : Machine
«flowPort» part : part
«flowPort» part2 : part

Figure 3: BDD of the flow shop system.

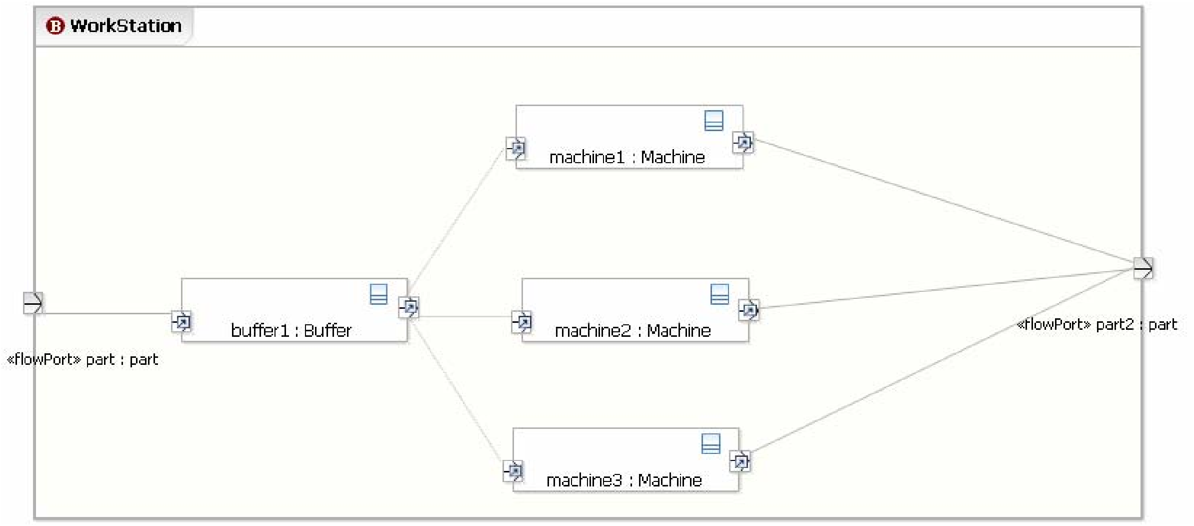


Figure 4: IBD of the workstation block.

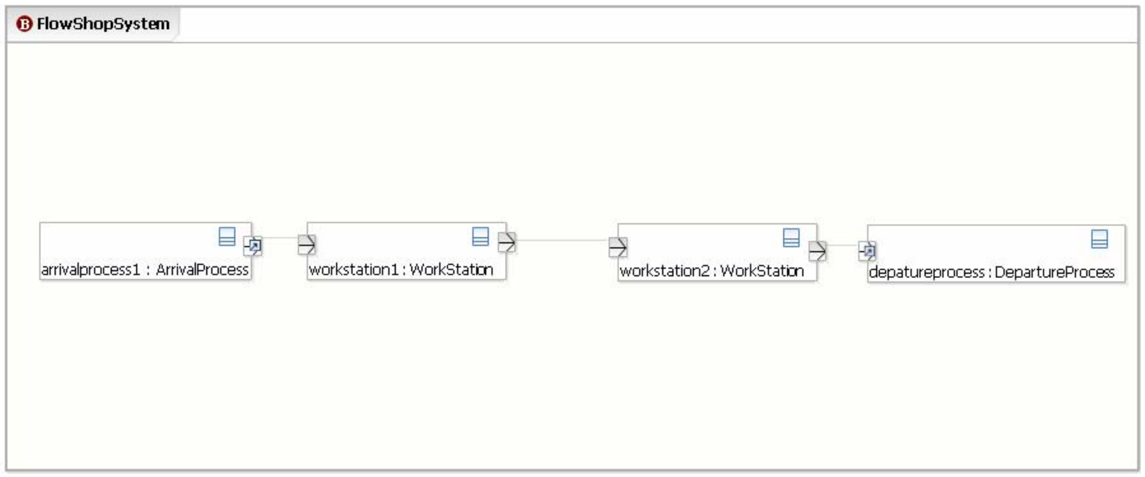
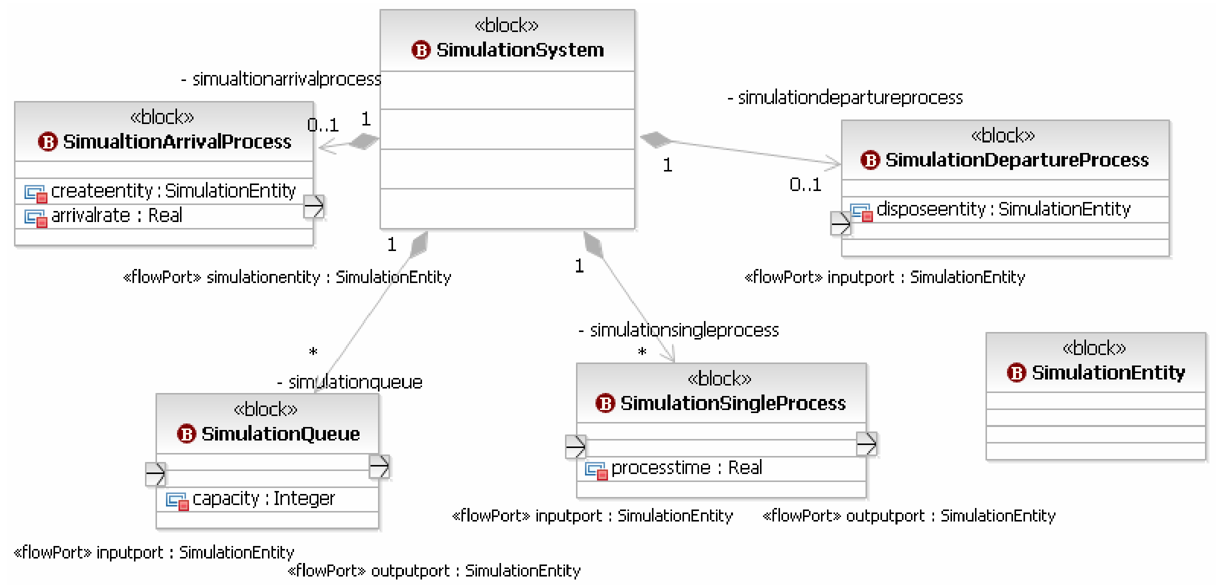
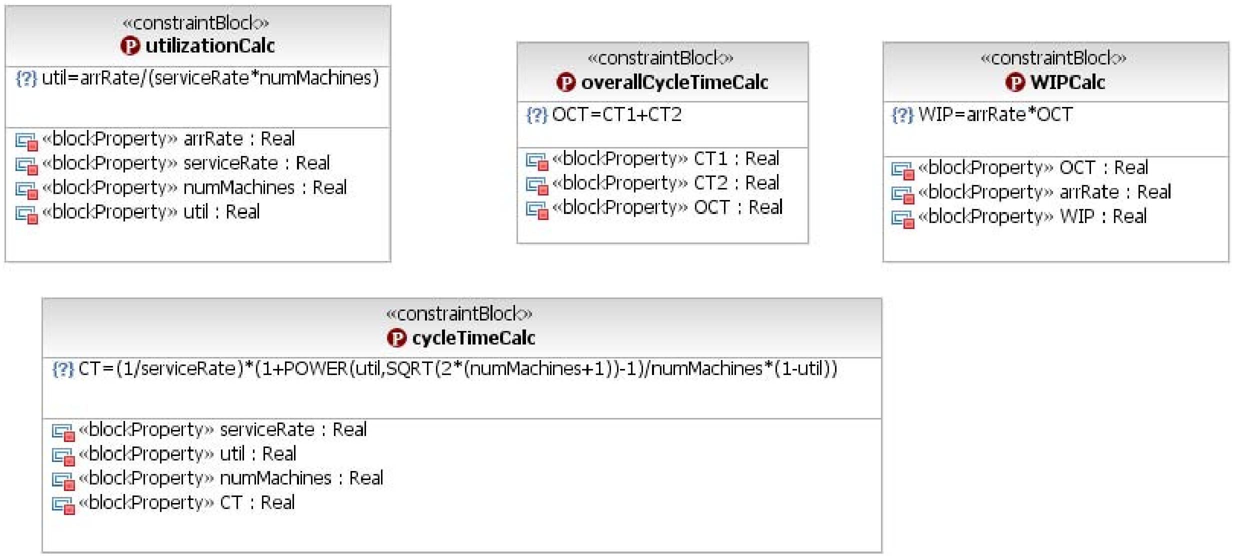

FlowShopSystem
arrivalprocess1 : ArrivalProcess
workstation1 : WorkStation
workstation2 : WorkStation
depatureprocess : DepartureProcess

Figure 5: IBD of the flowshop system block.


«block»
SimulationSystem
- simualtionarrivalprocess
0..1
1
«block»
SimualtionArrivalProcess
createentity : SimulationEntity
arrivalrate : Real
- simulationdepartureprocess
1
0..1
«block»
SimulationDepartureProcess
disposeentity : SimulationEntity
«flowPort» inputport : SimulationEntity
1
1
«flowPort» simulationentity : SimulationEntity
- simulationsingleprocess
*
*
- simulationqueue
«block»
SimulationQueue
capacity : Integer
«block»
SimulationSingleProcess
processtime : Real
«block»
SimulationEntity
«flowPort» inputport : SimulationEntity
«flowPort» outputport : SimulationEntity
«flowPort» inputport : SimulationEntity
«flowPort» outputport : SimulationEntity
Figure 6: Simulation meta-model.


«constraintBlock»
utilizationCalc
{?} util=arrRate/(serviceRate*numMachines)
«blockProperty» arrRate : Real
«blockProperty» serviceRate : Real
«blockProperty» numMachines : Real
«blockProperty» util : Real
«constraintBlock»
overallCycleTimeCalc
{?} OCT=CT1+CT2
«blockProperty» CT1 : Real
«blockProperty» CT2 : Real
«blockProperty» OCT : Real
«constraintBlock»
WIPCalc
{?} WIP=arrRate*OCT
«blockProperty» OCT : Real
«blockProperty» arrRate : Real
«blockProperty» WIP : Real
«constraintBlock»
cycleTimeCalc
{?} CT=(1/serviceRate)*(1+POWER(util,SQRT(2*(numMachines+1))-1)/numMachines*(1-util))
«blockProperty» serviceRate : Real
«blockProperty» util : Real
«blockProperty» numMachines : Real
«blockProperty» CT : Real
Figure 7: Queuing meta-model.

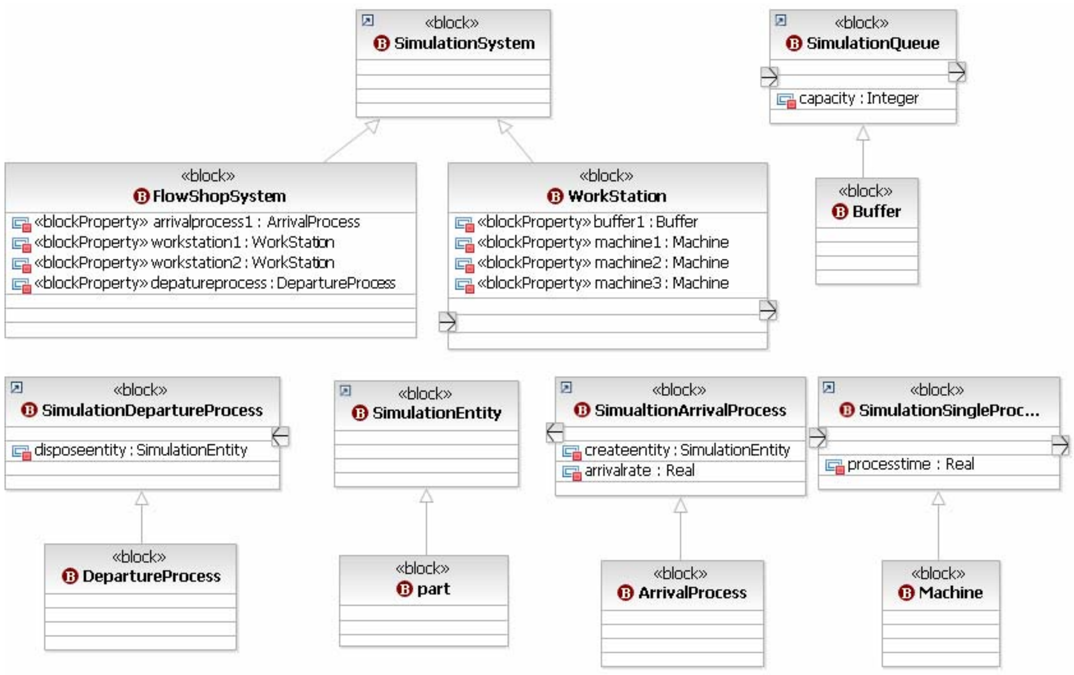

«block» SimulationSystem
«block» SimulationQueue
capacity : Integer
«block» FlowShopSystem
«blockProperty» arrivalprocess1 : ArrivalProcess
«blockProperty» workstation1 : WorkStation
«blockProperty» workstation2 : WorkStation
«blockProperty» depatureprocess : DepartureProcess
«block» WorkStation
«blockProperty» buffer1 : Buffer
«blockProperty» machine1 : Machine
«blockProperty» machine2 : Machine
«blockProperty» machine3 : Machine
«block» Buffer
«block» SimulationDepartureProcess
disposeentity : SimulationEntity
«block» SimulationEntity
«block» SimualtionArrivalProcess
createentity : SimulationEntity
arrivalrate : Real
«block» SimulationSingleProc...
processtime : Real
«block» DepartureProcess
«block» part
«block» ArrivalProcess
«block» Machine

Figure 8: BDD of the domain model mapped to the simulation analysis meta-model.

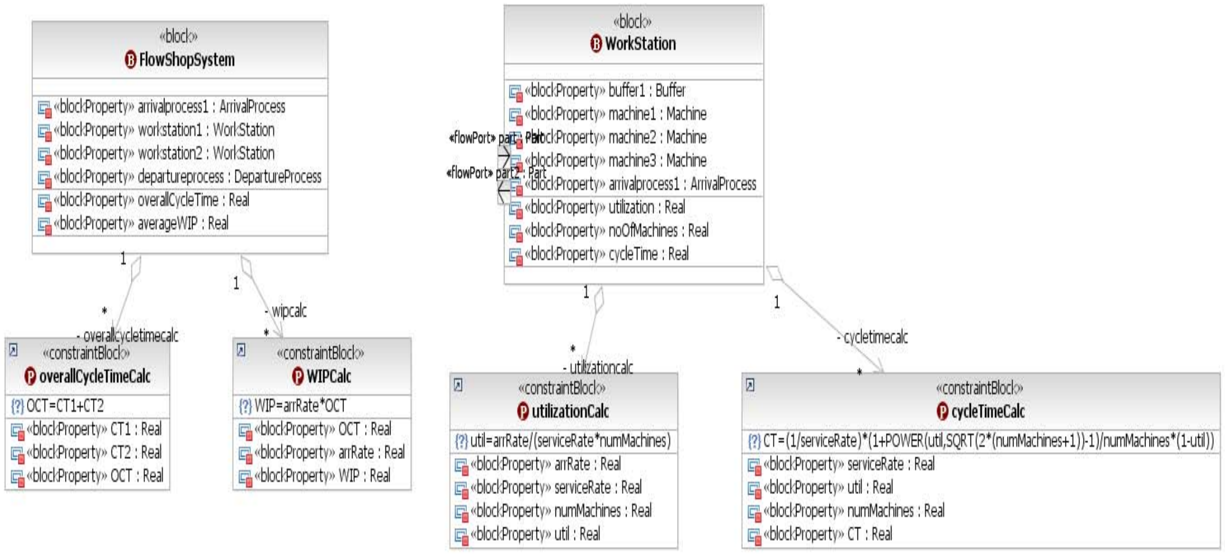

«block» FlowShopSystem
«blockProperty» arrivalprocess1 : ArrivalProcess
«blockProperty» workstation1 : WorkStation
«blockProperty» workstation2 : WorkStation
«blockProperty» departureprocess : DepartureProcess
«blockProperty» overallCycleTime : Real
«blockProperty» averageWIP : Real
1
*
- overallcycletimecalc
1
- wipcalc
*
«constraintBlock» overallCycleTimeCalc
{?} OCT=CT1+CT2
«blockProperty» CT1 : Real
«blockProperty» CT2 : Real
«blockProperty» OCT : Real
«constraintBlock» WIPCalc
{?} WIP=arrRate*OCT
«blockProperty» OCT : Real
«blockProperty» arrRate : Real
«blockProperty» WIP : Real
«block» WorkStation
«flowPort» part1 : Part
«flowPort» part2 : Part
«blockProperty» buffer1 : Buffer
«blockProperty» machine1 : Machine
«blockProperty» machine2 : Machine
«blockProperty» machine3 : Machine
«blockProperty» arrivalprocess1 : ArrivalProcess
«blockProperty» utilization : Real
«blockProperty» noOfMachines : Real
«blockProperty» cycleTime : Real
1
*
- utilizationcalc
1
- cycletimecalc
*
«constraintBlock» utilizationCalc
{?} util=arrRate/(serviceRate*numMachines)
«blockProperty» arrRate : Real
«blockProperty» serviceRate : Real
«blockProperty» numMachines : Real
«blockProperty» util : Real
«constraintBlock» cycleTimeCalc
{?} CT=(1/serviceRate)*(1+POWER(util,SQRT(2*(numMachines+1))-1)/numMachines*(1-util))
«blockProperty» serviceRate : Real
«blockProperty» util : Real
«blockProperty» numMachines : Real
«blockProperty» CT : Real

Figure 9: BDD of the domain model mapped to queuing analysis meta-model.

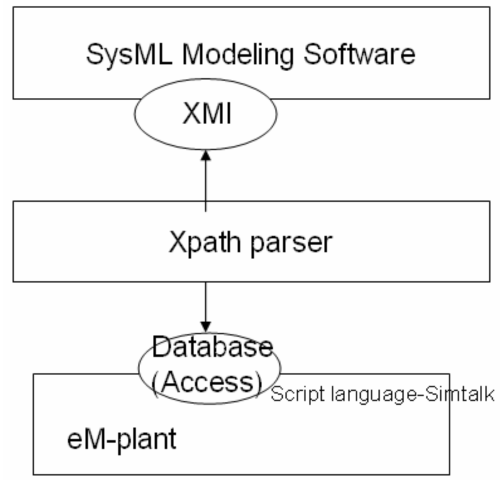


Figure 10: Architecture of the integration system.

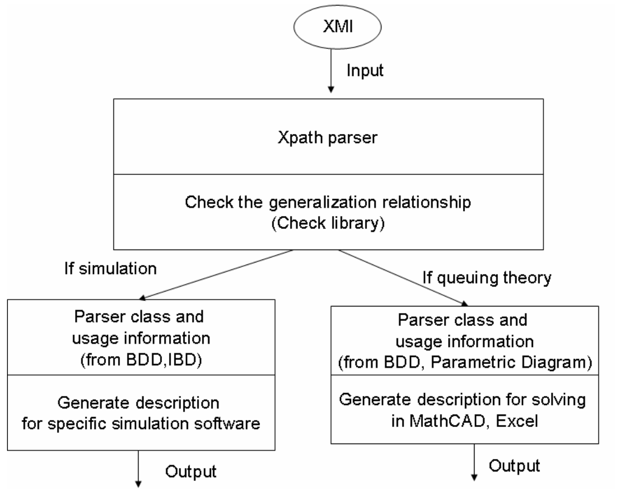

XMI
Input
Xpath parser
Check the generalization relationship (Check library)
If simulation
If queuing theory
Parser class and usage information (from BDD,IBD)
Generate description for specific simulation software
Parser class and usage information (from BDD, Parametric Diagram)
Generate description for solving in MathCAD, Excel
Output
Output

Figure 11: Logic of the integration system.

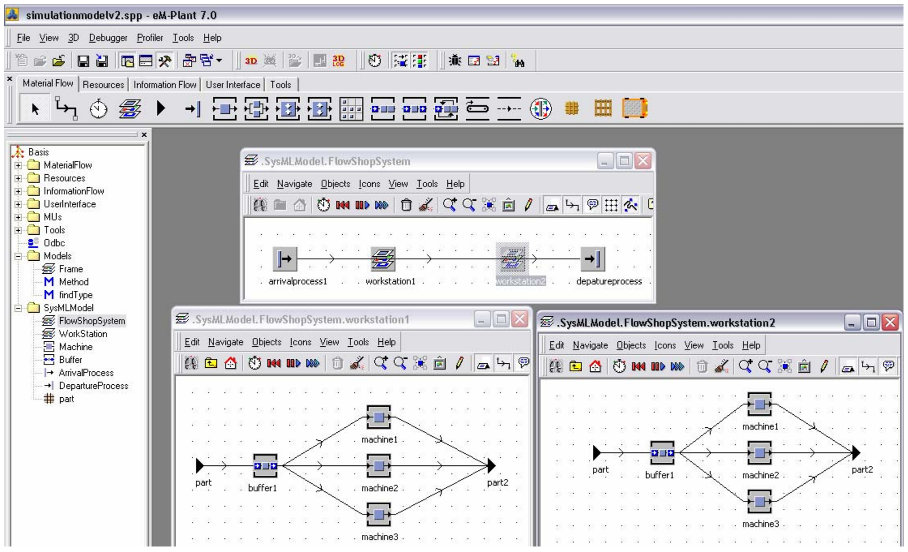


Figure 12: Screenshot of the automatically generated flowshop system simulation model.

REFERENCES

EmbeddedPlus©. *SysML ToolKit* 2006. Available via <www.embeddedplus.com/>.

IBM. 2004. Rational software delivery platform.

Law, A. M. and W. D. Kelton. 2000. *Simulation modeling and analysis*. 3rd ed. Boston: McGraw-Hill.

OMG. 2006. SysML final adopted specification 2006. Available via <www.omg.org/cgi-bin/doc?ptc/06-05-04>.

OMG. 2007. UML 2007. Available via <www.uml.org/> [accessed April 9, 2007].

UGS™. 2003. eM-Plant.

W3C. 2007. XML path language (XPath) version 1.0 1999. Available via <www.w3.org/TR/xpath> [accessed April 10, 2007].

AUTHOR BIOGRAPHIES

EDWARD HUANG is a Ph.D. candidate in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. His research interests include simulation modeling and , as well as application of simulation in semiconductor manufacturing. His email address is <edwardhuang@gatech.edu>.

RANDEEP RAMAMURTHY is a Masters student in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. His research interests include systems modeling and analysis of industrial engineering systems. His email address is <randeep@gatech.edu>.

LEON MCGINNIS is Gwaltney Professor of Manufacturing Systems at Georgia Tech, where he also serves as Director of the Product and Systems Lifecycle Management Center, Associate Director of the Manufacturing Research Center, and Director of the Keck Virtual Factory Lab. His research is focused on the representation of complex industrial systems, such as warehouses and factories, to enable analytic and simulation modeling to support performance assessment, behavioral prediction, and system design. His email address is <leon.mcginnis@gatech.edu>.