Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

SYSTEM MODELING IN SYSML AND SYSTEM ANALYSIS IN ARENA

Ola Batarseh
Leon F. McGinnis

The School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA, USA

ABSTRACT

A Model Driven Architecture approach is employed to support the practice of discrete-event simulation. OMG's System Model Language, OMG SysML™, is used to define a platform independent model (PIM) and auto-translate it into an appropriate platform specific model (PSM). The implementation and the nature of the transformation from PIM to PSM are clearly addressed to enable: (i) formal modeling of systems using their own semantics in SysML, (ii) SysML model verification and validation by stakeholders, (iii) automatic translation of system models expressed in SysML into analysis models as the PSM, and (iv) maintainability of this approach to accommodate system changes and extensions very easily. The proposed approach can be used for any analysis tool and application domain. In this paper, we choose to model transaction-based examples elicited from the manufacturing domain in SysML and translate them into Arena™ models using the Atlas Transformation Language.

1 INTRODUCTION

Discrete-event simulation (DES) tools are used to assess complex systems when analytical approaches are not possible. Therefore, the systems that are usually the good candidates for DES analysis are complex with multiple stakeholders. These stakeholders use *ad-hoc* methods to conceptually model their systems and transfer their inputs to the simulation analysts to understand the system scope for analysis. These *ad-hoc* methods have no standards or structure to follow and can take various forms such as documents, diagrams, databases, etc. We propose using a model-based system engineering approach to support the modeling of systems for DES analysis. Our motivation is to eliminate the use of the *ad-hoc* methods to model the system, and to reduce the cost, time, and efforts in building the simulation models.

Using *ad-hoc* methods to represent the system of interest impacts the fidelity of the communication between the stakeholders which may introduce doubt as to whether the simulation analysts have grasped fully the intent of the stakeholders. In order to ensure that the simulation analysts receive the right information, significant time and effort are consumed in this phase of any simulation project. Moreover, the informality of these methods hinders the re-usability of the system descriptions or investigating automatic model transformations. Modeling systems in this way is an arcane task for all the stakeholders' involved.

After all the efforts consumed to transfer the required system knowledge to the simulation analysts, the simulation models are manually coded based on the analysts' acquired knowledge. The manual efforts for coding may take many man-hours, and the associated cost sometimes is a reason to avoid using DES analysis. Simulation analysts use some *ad hoc* approaches to save time in building the simulation models such as copying and pasting with some modifications, or perhaps by developing templates, libraries, or simulators. Furthermore, the validation of these simulation models can only be done directly by the simulation analysts as they are the experts in this area. The process owners will only be able to compare the

simulation model results to the anticipated values for verification purposes; to them, the simulation model is a black box.

This paper addresses a solution for these problems faced in the practice of DES. This solution proposes developing the conceptual models between the stakeholders using a formal application domain language, referred to as a *domain-specific language (DSL)*. The System Modeling Language (SysML) is customized to create this DSL, and also to develop a meta-model for the target simulation solution (Weyprecht and Rose 2011). This solution is based on the Model Driven Architecture framework (http://www.omg.org/mda/). It is initiated with a platform independent model which is translated into a platform specific model—a simulation model—using transformation technologies. The mapping between these models is essential for realizing the transformation.

Prior research has established the feasibility of applying the MDA approach to the creation of simulation models (McGinnis and Ustun 2009, Batarseh and McGinnis 2012). The essential concept is to create a DSL for a domain of application in SysML e.g., electronics assembly, warehousing, or supply chain management, and create a model transformation to a particular kind of analysis model, e.g., discrete event simulation. The system of interest is described using the DSL, and the resulting model is transformed automatically to an analysis model. What has not been adequately addressed are the fundamental issues associated with creating the DSL and the transformation, so this paper address the question "How should one set about to actually deploy this methodology using SysML as the formal modeling language?"

This paper is concerned with discrete event simulation as the analysis tool of interest; nevertheless, the proposed framework can be applied to other analysis solutions such as optimization, queueing networks, and financial models, etc. This paper uses Arena as the DES tool of interest. Furthermore, Atlas transformation language (ATL) is used in this work to establish the transformation from SysML to Arena. The manufacturing domain is used to illustrate the implementation of the proposed approach.

The remainder of this paper is organized as follows. Section 2 reviews related topics. In Section 3, we describe our proposed approach to enable the auto-generation of simulation models to Arena. In Section 4, we present the description of the analysis tool profile. In Section 5, we explain the use of this profile to build model libraries as the problem DSL. Section 6 illustrates the use of transformation technologies to pertain the desired transformation. Finally, Section 7 concludes this paper.

2 BACKGROUND

In this section, we introduce SysML as the desired modeling platform for a DSL to create the PIMs, and the technology for the transformation of the PIMs into PSMs.

2.1 System Modeling Language

The Unified Modeling Language (UML) is an industry standard for a general purpose modeling language for object-oriented software development, maintained by the Object Management Group, OMG (http://www.uml.org/). OMG has created the System Modeling Language, SysML, standard as an extension of UML to support modeling of complex systems that involve humans and hardware as well as software components (http://omgsysml.org/). SysML reuses a subset of UML and provides additional modeling capabilities for requirements and parametric relationships, and augments the UML activity, block definition, and internal block diagrams. While it is a formal language, conforming to Meta-Object Facility MOF, it has a graphical user interface, making most diagrams relatively easy to understand.

SysML has a growing user base, largely in the aerospace and defense industries (Gedo 2012). SysML offers two possible ways to create DSLs--model libraries and profiling (Selic 2007). A model library contains a set of reusable model constructs for a given domain. Profiling extends the SysML language using stereotypes that add new reusable language concepts in a manner similar to the way SysML is developed from UML. Both mechanisms to customize the language are valid and have been used in a number of applications. They may be combined, i.e., first extend the language with a profile and then create library objects using the language extension. However, each approach supports DSL development in a different

way and adopting one over the other is a significant decision because of its impact on development of model transformations that translate the problem statement in the DSL into an analysis model which will be solved using a specific solver technology. However, these efforts have varied in their approach.

Adapting MDA to manufacturing systems would require a DSL with the manufacturing system semantics, such as, parts, operator or machine, inspection process etc. We have used this approach in the past to implement a DSL for a relatively large electronic assembly system (Batarseh and McGinnis 2012) as a profile in SysML. This approach has offered significant savings when used in practice, nevertheless, it has some shortcomings; in particular, if the application domain is extended by adding additional semantics to the DSL, the transformation script must be revised, which can be a significant task. The proposed approach overcomes this shortcoming as will be explained.

2.2 Model Transformation

Transformation technologies enable the reuse of knowledge captured in the DSL to generate analysis models. Czarnecki and Helsen (2006) presented a metamodel-based transformation framework, as in Fig. 1. This framework is the most often used for model transformations to support MDA (OMG, MDA Guide Version 1.0.1, 2003).

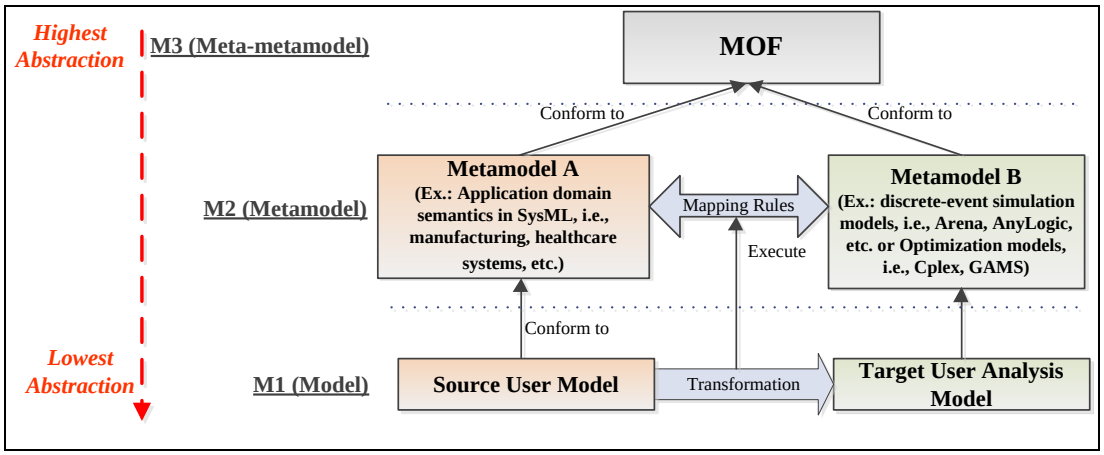

Fig. 1. Metamodel-based Transformation Framework

This framework employs a model:meta-model concept: every model conforms to a meta-model, except for MOF, which conforms to itself (http://www.omg.org/mof/). There is a single meta-meta-model (or meta-meta language), which is MOF, and every meta-model conforms to this meta-meta-model. The conforming meta-models, M2 in Fig. 1 may define, for example a domain specific language, such as manufacturing semantics on the source side using SysML language, or an analysis language, such as Arena, on the target side.

The mapping rules between the source and target meta-models enable the translation of the source user model into the target user analysis model, and are implemented as a *mapping model*, or *script*. Developing the script requires a deep understanding of both the domain and analysis meta-models. The benefits of this approach are numerous, such as portability, gain in productivity, and interoperability (Kleppe, *et al* 2003). Model transformation tools have demonstrated the feasibility and applicability of the approach. Examples of the model transformation tools are: ATLAS (ATLAS Group 2007), MTF (IBM 2009) and Semaphore (SINTEF 2006) implemented using the Eclipse platform (http://www.eclipse.org). Each of the transformation tools has its strengths and weaknesses in application (Ben Salem, *et al* 2008). This paper uses ATLAS for transformation purposes.

3 METHODOLOGY: PROFILE FOR TOOL AND MODEL LIBRARIES FOR PROBLEM DOMAIN

As mentioned earlier, SysML offers two ways to create DSLs, i.e., profiles and libraries. Our approach uses the two means to create DSLs. The profiling mechanism is exploited to create a specific analysis tool profile which is , in effect, an Arena DSL referred to as SysML4Arena. Using this profile, model libraries are employed to build the application domain semantic which is the manufacturing domain in this paper. However, this proposed approach can be adapted for any analysis tool and application domain.

It is crucial to distinguish the different kinds of expertise required to support simulation modeling: (i) domain expertise, (ii) modeling expertise, and (iii) analysis expertise. The domain experts are the process owners who use domain semantics to describe their systems. Modeling expertise translates the system description using domain semantics into a form usable by the analysis experts, often using modeling tools that are diagrammatic. The analysis expertise is required for building and using the analysis models. Each of these experts contribute for a specific task in the proposed approach. Fig. 2 shows an activity diagram in SysML to explain the roles of the experts involved.

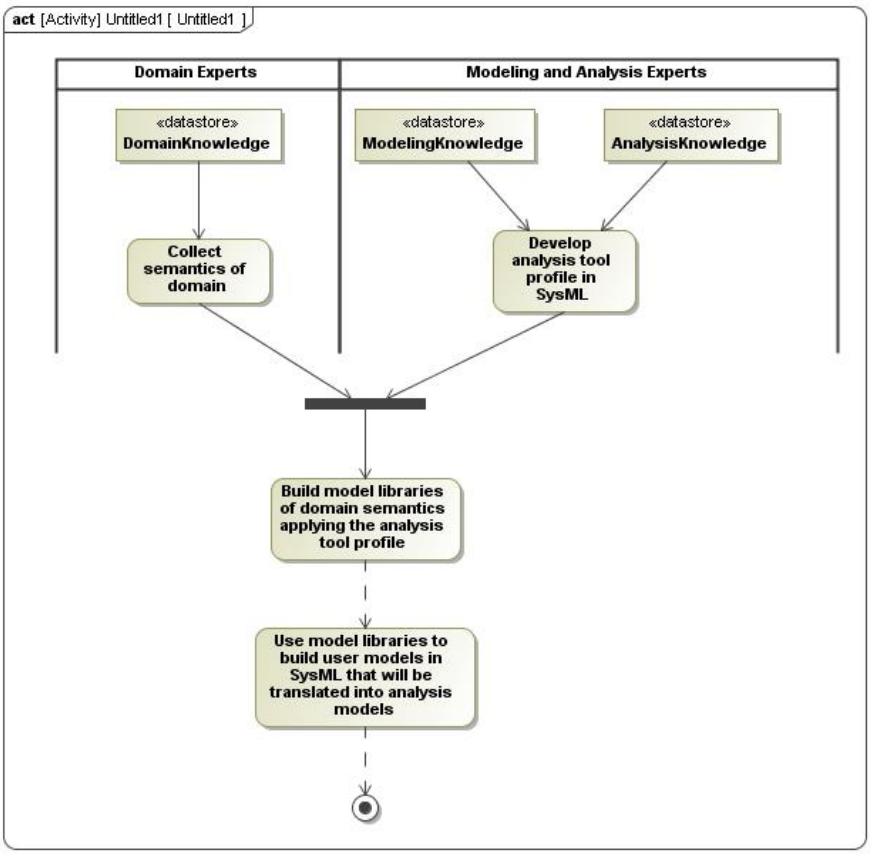

Fig. 2. Activities involved for implementing the proposed framework in SysML to support the auto-translation of SysML models into simulation models

The domain experts in a manufacturing domain would provide related semantics such as *parts, machines, operators, processes, etc.* The modeling experts translate these semantics into diagrams as *Bill of Materials* and *process plans* to visually conceptualize the system requirements and specifications. These diagrams are the main means of communication between the system modelers and the simulation analysts. The proposed approach captures the domain semantics in a model library using *Blocks.* The library blocks use SysML4Arena profile to model their necessary *activities* such the process plan of a manufac-

turing part. The following section explains SysML4Arena which also illustrates its usage to model specifically *activities.*

The result of this proposed approach is to provide the modeling and analysis experts one formal platform in SysML to communicate the domain semantics provided by the domain experts. Using SysML for this purpose replaces all the various *ad-hoc* methods to establish the communication between the stakeholders. In addition, the final result is domain model libraries that the end user will use to build formal SysML models based on the *DSL*. The model library blocks uses the analysis tool profile stereotypes to model the details of the semantic it represents; the resulting model will be auto-translated into the analysis tool. Finally, this approach can be applied to any domain application such as healthcare, warehousing, computer networks, etc. that uses Arena for DES analysis.

4 THE SYSML PROFILE FOR THE ANALYSIS TOOL, SYSML4ARENA

A SysML profile is a set of stereotypes that define how the SysML meta-classes are extended to define the semantics and syntax of the modeled domain. Our proposed approach employs the profiling mechanism to develop the analysis tool constructs using stereotypes. The resulting SysML profile for the analysis tool is a high-level language that can be used to develop model libraries representing the application domain semantics. Herein, we discuss the developed profile for Arena, SysML4Arena, which is implemented by the efforts of the modeling and the analysis experts. The analysis experts provide the analysis tool semantics and the modeling experts decide on which meta-classes to extend for the stereotypes.

Arena is a process-oriented modeling tool for discrete-event systems. In other words, the modeling in Arena environment is structured as a workflow of stepwise activities and actions. Therefore, the *activity diagram* in SysML is the appropriate way to develop the Arena stereotypes, using activity diagram components' meta-classes. Fig. 3a depicts a simple model in Arena in which an "Entity" gets created from the "Create" module in Arena, it gets delayed for processing using the "Process" module that seizes a machine modeled as a "Resource". Fig. 3b depicts a SysML activity diagram that models the same system to highlight the compatibility of Arena models with the activity diagrams in SysML. Besides the process flows that are modeled in Arena, there are other modules that model the objects of the system such as "Entity", "Resource", etc. The objects are best modeled in SysML as *Blocks*.

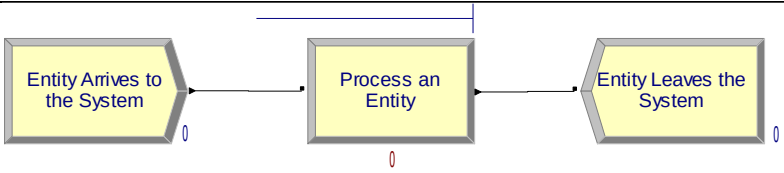

Fig. 3a: Simple model in Arena

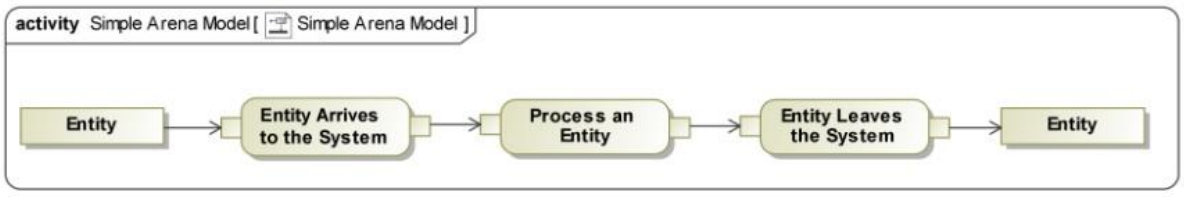

Fig. 3b: Modeling Arena Simple Model in SysML using an Activity diagram

The SysML4Arena profile consists of stereotypes that capture the Arena tool constructs, as shown in Fig. 4. The meta-classes that are used to extend SysML4Arena stereotypes are mainly:

1. Actions: to extend any process in Arena, such as, Create, Process, Delay, Dispose, Transfer, etc.
2. Object Flows: to extend the Connections between the processes in Arena.

3. Blocks: to extend the object modeled in Arena, such as, Entity, Resource, Schedule, etc.

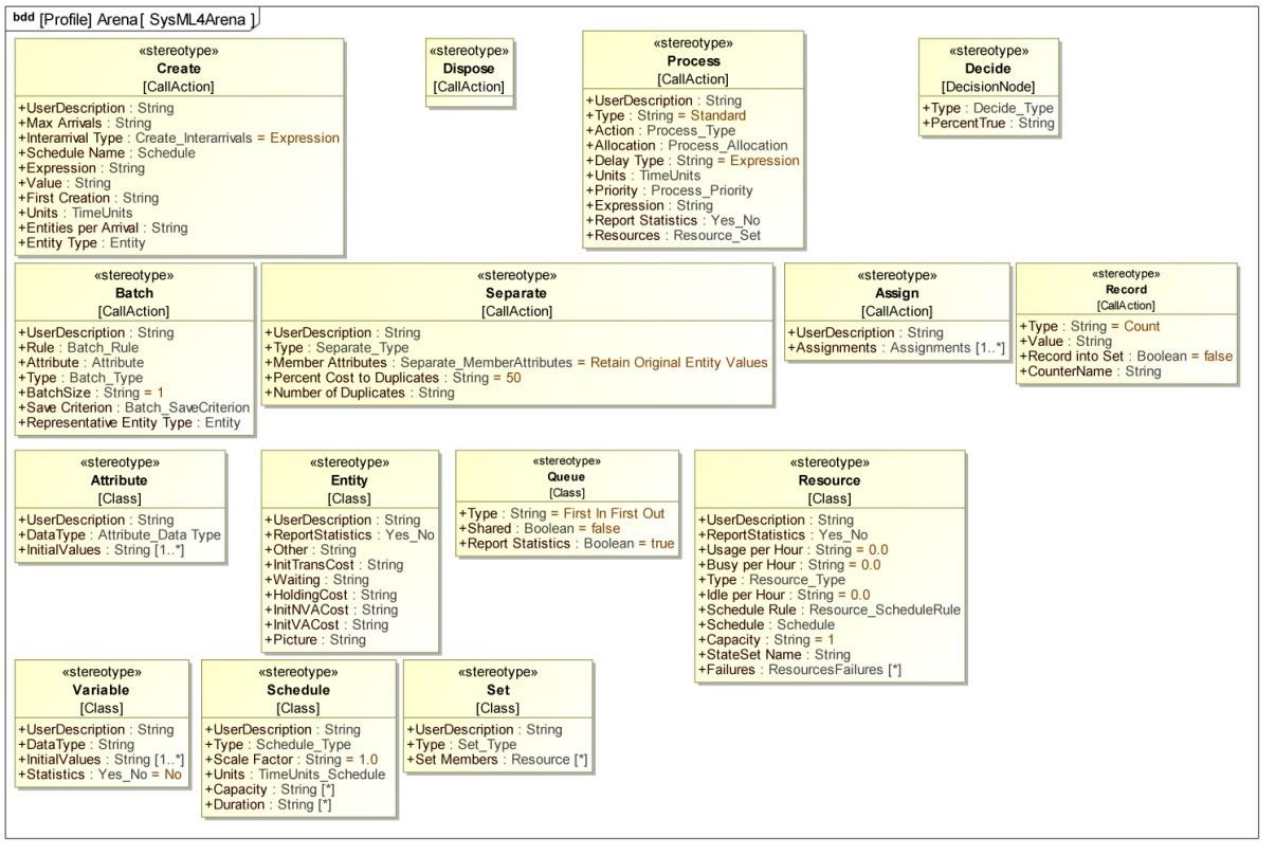

Fig. 4. SysML4Arena Profile in SysML that implements the Basic Process Template as stereotypes

This profile is used to construct the application domain model libraries. The following section discusses in details the development of domain model libraries using analysis tool profile, i.e., SysML4Arena.

5 DOMAIN MODEL LIBRARIES IN SYSML

As mentioned earlier, the analysis tool profile is used to build model libraries. The analysis experts are primarily responsible for creating these model libraries. The Arena simulation environment is modeled in the SysML tool to simplify the analysts' use of SysML4Arena to create the domain specific modules using Arena semantics. Fig. 5 shows customized SysML4Arena menus in SysML to ensure that the Arena simulation experts do not find difficulty in building the domain model libraries. This implementation is in MagicDraw's SysML editor.

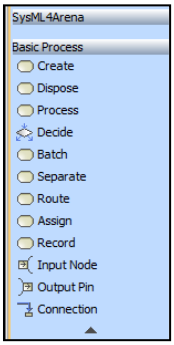

Fig. 5. Customized menus for SysML4Arena in MagicDraw

The simulation analysts would not need a deep knowledge of SysML semantics in order to build model libraries. Model libraries are built using *Blocks* to allow adding properties, such as, *value property, part property, etc.* The value properties are defined for the end users to characterize the data instances of the model. Two examples from manufacturing are presented to illustrate the implementation of model libraries in SysML using SysML4Arena.

Example 1: One of the domain semantics of the manufacturing system is Operator. An operator is one resource that is needed for a process to perform a specific task in a part's process plan. The simulation analyst translates the operator semantic as a *Resource* in the simulation model. In other words, the *operator* in the domain semantics is matched to a *resource* in the Arena semantics. Fig. 4 shows that the stereotype Resource is extended from a Block. In addition, this block has many attributes that are added to reflect parameter values for Resource in Arena. For example, some of these attributes include *Capacity, Failures, Schedule, etc.*

The simulation analyst creates a Block named *Operator* in the model library to implement the domain semantics and stereotypes it as a <<Resource>>. This implementation allows the end user to use domain semantics in SysML, i.e. Operator, and ensures the mapping with the analysis tool semantics by stereotyping it, i.e., Resource, for translation purposes. The required inputs from the end user are added as properties to the library block, i.e. Capacity of the Operator. Fig. 6 shows a containment tree in a SysML project implemented in MagicDraw to illustrate creating domain model libraries using this approach.

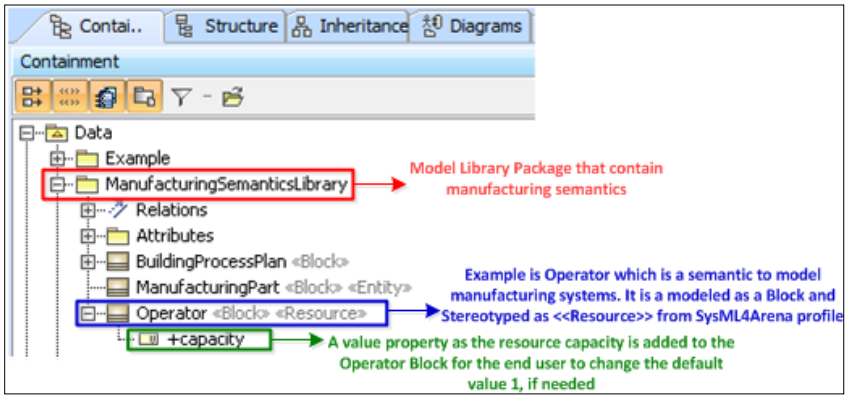

Fig. 6. An Operator: Example in Manufacturing Model Libraries

The end user that builds SysML models for simulation purposes uses domain semantics from this model library to build the SysML model, as Fig. 7 illustrates. An example is shown for two Operators: (i) Assembly Machine Operator with a capacity equal to 1, and (ii) Thermal Testing Operator with a capacity equal to 3. Using this approach the end user is required to understand neither SysML semantics nor Arena semantics.

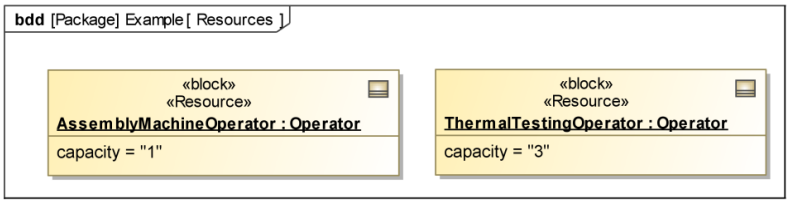

Fig. 7. Two instances of the Operator Semantic Modeled by the End User of Domain Library

Example 2: One of the domain semantics of the manufacturing system is manufacturing process. Some examples of these processes are *Build, Assembly, Inspection, Rework, Troubleshoot, Testing, etc.* Here, we give an example of an Assembly process in manufacturing. Assume that an Assembly process in

a manufacturing system is essentially a process that is executed on a batch of parts. A simulation analyst using Arena to translate this logic uses two Arena modules to model an Assembly process, which are:

1. Batch: This module allows the user to input a desired Batch Size for the process.
2. Process: This module essentially does the processing on the batch by delaying the batch according to the input processing time and seizing a resource for processing.

Our proposed approach let the simulation analyst to build this logic in SysML using Arena semantics, i.e., stereotypes in Fig. 4 customized as in Fig. 5. The implementation of this Assembly Process is explained as follows. The simulation analyst's objective is to implement the logic for the modeled process and provide the end user with the domain semantics for modeling. The model library elements are mainly Blocks, and a Block can own a specific Activity diagram that models a process. Therefore, the simulation analyst will create a Block and rename it from the domain semantics, i.e., Assembly, and add an activity diagram to model the process logic using Arena semantics, as shown in Fig. 8.

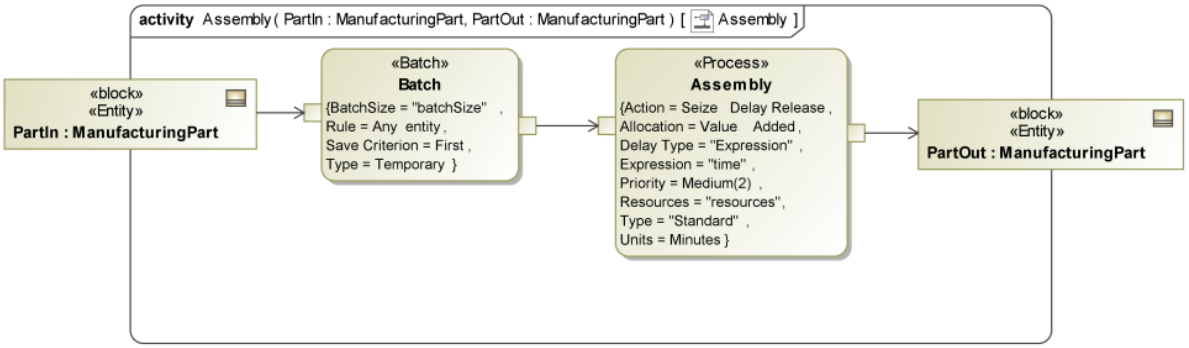

Fig. 8. An Assembly Process: Example in Manufacturing Model Libraries

The simulation analyst will add properties to the Assembly Block to allow the end user to input required data. Examples of these inputs for the assembly process are: batch size, processing time after batching, and resources. Fig. 9 exemplifies these properties on the Assembly Block.

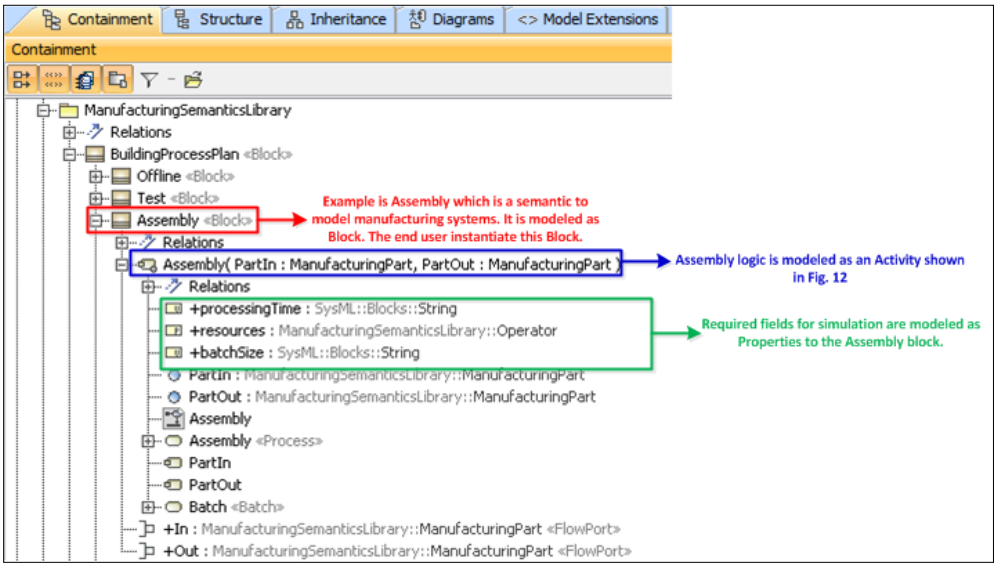

Fig. 9. Assembly Process Properties

The end user will use the Assembly semantics from the library to model a part or product's assembly process. The process plan itself is identified as a Block, and the details of the process are defined using the *internal block diagram (IBD)*. Fig. 10 depicts a process plan as an IBD, where a sub-component is received then there is an assembly process to produce a final product. The value properties defined for the assembly block are stored as slots of the instance. Fig. 10 also highlights the slots of the assembly process. Again, the objective is to highlight how the end user will be dealing only with the application semantics to build user models.

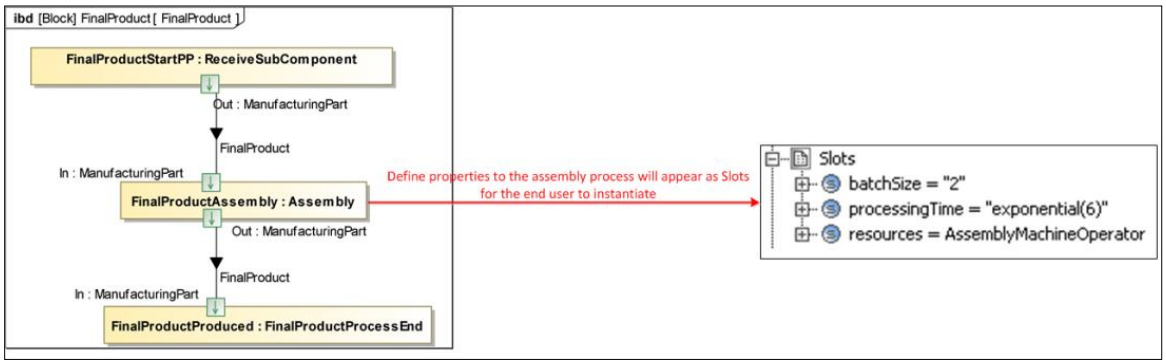

Fig. 10. Building Process Plan for a Part as an IBD and instantiating required input fields

6 MODEL TRANSFORMATION

Model transformation executes mapping rules to automatically translate a model from one software environment to another. The objectives of the models might be different. In this paper, a SysML model based on DSL for system modeling is translated into a simulation model for analysis. The ATLAS transformation language is used for the mapping rules and Eclipse (JAVA Editor) is the platform used to execute the transformation.

The SysML user model to be translated is constructed using instances of model library blocks that represent the DSL semantics. These blocks are stereotyped with the Arena semantics as explained above and are referred to as "Classifiers" of the instances. The mapping rules are written at the level of the user model components, i.e., the instances used from the model libraries. The mapping rules match these instance specifications with Arena semantics according to the applied stereotype of its classifier. For instance, AssemblyMachineOperator is an instance of the Operator which is considered its classifier.

In order to execute the transformation the transformation engine requires several input files. In this configuration, one of the inputs to the transformation engine is the SysML model's exported XMI files representing the SysML meta-model, the SysML4Arena profile, the DSL model library, and the user model. The relationships between these inputs are also realized within these files. The target meta-model is also an input to the transformation engine. This ensures that the output XMI file from the transformation conforms to the target meta-model. The framework of the transformation is captured in Fig. 11.

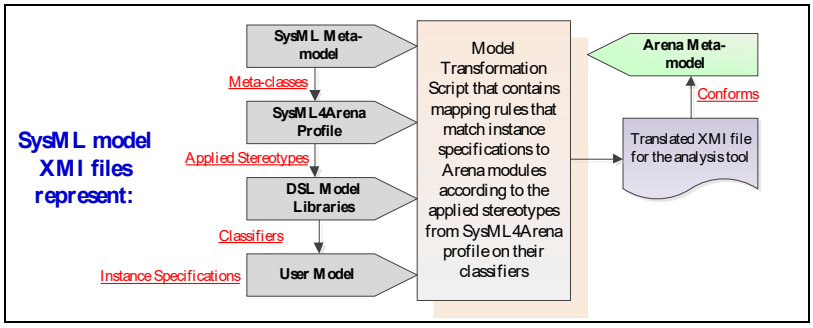

Fig. 11. Model Transformation Framework: Inputs and Output

An example of a mapping rule in a transformation script that matches an instance specification of the Operator with an applied stereotype <<Resource>> is shown in Fig. 12. Fig. 6 shows the part of the model libraries where the Operator is modeled as a Block and stereotyped as <<Resource>>. The translator searches for all Block instances that are stereotyped as a <<Resource>> from SysML4Arena profile and translates them into a Resource module in the generated XML file.

There are 4 highlighted sections in the mapping rule shown in Fig. 12:

(1) In the beginning of the rule, the source model element that will be matched is specified. The "from" is the keyword that states the source model element. In this implementation, the model element is an instance specification identified as "uml!InstanceSpecification".

(2) The second section places conditions on this instance specification. This rule is to match the instance specification whose classifier is stereotyped by a <<Resource>> from the SysML4Arena profile. The if-statement above executes this logic to match only the instance specification with this condition.

(3) Here, the types of the generated target model elements are specified from Arena meta-model. The target module is specified as *Arena!"BasicProcess_x007C_Resource"*. In result, this rule generates a resource model in Arena for every instance specification selected in (1) with the condition met in (2).

(4) The rule states the way these target model elements must be initialized from the matched source elements. The *Capacity* of the resource is the only value property the end user specified. The other fields were not required from the end user and kept their default value from Arena. In addition, the value initialized by the end user to specify the capacity of the instance specification is entered as a *slot* value to this instance. The logic associated with the capacity searches for the slot with the defining feature *capacity* and returns the value specified by the end user.

```
rule umltoResource {
        from
                dt: uml!InstanceSpecification ..... (1)
                (
                        dt.classifier->iterate(e; result: Boolean = false|
                        if (not e. getAppliedStereotype('SysML4Arena::Resource'). oclIsUndefined())
                                then true
                                        else false
                                                endif) .....(2)
                )
        to
                out: Arena!"BasicProcess_x007C_Resource" .....(3)
                (
                        SerialNumber <- dt. name. hashCode(). toString(),
                        ModelLevelID <- '1',
                        X <- '0',
                        Y <- '0',
                        Name <- dt. name,
                        UserDescription <- '',
                        ReportStatistics <- 'Yes',
                        Usage <- '0.0',
                        Busy <- '0.0',
                        Type <-    'Capacity',
                        Idle <- '0.0',
                        ScheduleRule <- 'Wait',
                        Schedule <- '',
                        Capacity <-dt.slot->iterate(e; result: String = '' |
                                if (e.definingFeature.name='capacity')
                                        then e.value.at(1).value.toString()
                                                else ''
                                                        endif), .....(4)
                        StateSetN <- '',
                        InitState <- '',
                        FDM_x0020_Name <- '',
                        FDM_x0020_Id <- '0',
                        Arena_x0020_Imported_x0020_Name <- '',
                        Base_x0020_Efficiency <- '1.0',
                        Efficiency_x0020_Schedule <- ''
                )
}
```

Fig. 12. Mapping Rule Translates Instances of Operator to Resources in Arena

In this framework, the transformation rules are not connected to the domain semantics, i.e., Operator but to the <<Resource>> stereotype from SysML4Arena. This transformation is free from the DSL semantics which makes it feasible to have one transformation for any application domain that uses SysML4Arena profile to build its model library as illustrated in Fig. 13.

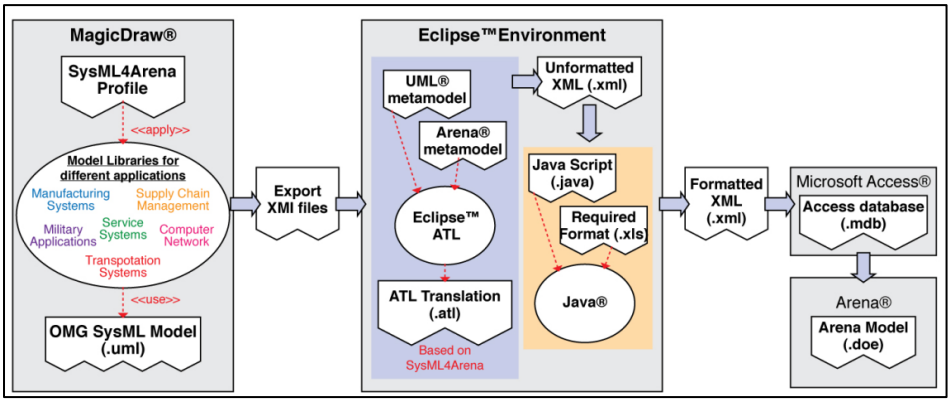

Fig. 13. Transformation from SysML to Arena

The proposed framework essentially transforms SysML models into simulation models in Arena. The SysML models are built using domain semantics that are tagged with Arena stereotypes collected in SysML4Arena profile. The transformation is enabled for any application domain that uses SysML4Arena. The transformation is performed in Eclipse environment: (i) ATL script translates SysML model into unformatted XML file, and (ii) Java script fixes this file into a formatted XML file. An Arena legacy is exporting only Access database files; this implies using Microsoft Access as an intermediary between the translated XML file and Arena.

7 SUMMARY AND FUTURE WORK

In this paper, we have demonstrated an MDA approach to model systems using domain-specific language in SysML. The domain semantics are components of the model library that use SysML4Arena profile to implement the required logic for building the analysis models in Arena for discrete-event simulation. The proposed approach automatically translates the SysML models into Arena models using ATLAS model transformation technology. The mapping rules of the transformation script are not restricted to the application domain semantics but to the analysis tool profile, SysML4Arena. The advantages of this approach are: system's specification formally modeled via SysML based DSL instead of *ad-hoc* methods; improved involvement of all the stakeholders in both the modeling efforts and in model verification and validation; automatic translation from domain model to analysis model eliminates the time and error of the manual analysis model coding.

The current capability of the proposed approach is limited to translating the SysML models into one DES software tool, Arena. In addition, it has been implemented for one application domain elicited from manufacturing systems. Some future work for this approach is to model two systems from different applications domains in SysML and use the same transformation script to obtain the DES models in Arena. Furthermore, it is interesting to explore the possibilities of translating the model library domain semantics into multiple analysis tools in DES and other analysis approaches such as optimization. MDA promises new approaches to model systems and execute the appropriate analysis.

REFERENCES

ATLAS group LINA & INRIA (2007). Atlas Transformation Language, User Guide. Downloaded from: http://www.eclipse.org/gmt/atl/doc/

Ben Salem R Grangel R Bourey JP (2008). A comparison of model transformation tools: Application for Transforming GRAI Extended Actigrams into UML Activity Diagrams, Computers in Industry, 59(7):682-693.

Batarseh O and McGinnis L (2012). SysML to Discrete-event Simulation to Analyze Electronic Assembly Systems. Accepted to appear at the 2nd International Workshop on Model-driven Approached for Simulation Engineering; Orlando, Fl.

Czarnecki K and Helsen S (2006). Feature-based survey of model transformation approaches. IBM Syst J 45(3): 621–645.

Gedo C (2012). DISA' s DoDAF V2 with Model Based Systems Engineering and Systems Modeling Language (MBSE)/Systems Modeling Language (SysML). Downloaded from: http://dodcio.defense.gov/sites/dodaf20/products/1320-1340_DISA_DoDAF_MBSE_SysML_Gedo_01-05-2012_V1.pptx

IBM (2009). IBM Model Transformation Framework 1.0.2 Programmer's Guide. Downloaded from: http://www.alphaworks.ibm.com/tech/mtf

Kleppe A Warmer J Bast W (2003) MDA Explained, The Model Driven Architecture: Practice and Promise, Addison-Wesley.

McGinnis, L. F., and Ustun, V., 2009, "A simple example of SysML-driven simulation," in Proceedings of the 2009 Winter Simulation Conference, Austin, TE, USA, pp. 1703–1710

Selic B (2007) A systematic approach to domain-specific language design using UML. In 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), P2–P9.

SINTEF (2006). Semaphore Guide. Downloaded from: http://www.modelbased.net/semaphore

Weyprecht, P., and Rose, O., 2011, "Model-driven Development of Simulation Solution based on SysML starting with the Simulation Core." In Proceedings of the Spring Simulation Conference

OLA BATARSEH is a Post-doctoral Research Fellow in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. She received his MS and PhD from the University of Central Florida in 2008 and 2010, respectively. She holds a B.Sc. in Industrial Engineering from the University of Jordan. Batarseh research focuses on reliable discrete-event simulation to support decision making. Her major research interests are in the areas of systems engineering, discrete-event simulation, model transformations, total input uncertainties, and imprecise probabilities.

LEON MCGINNIS is Professor Emeritus in the Steward School of Industrial and Systems Engineering at the Georgia Institute of Technology, where he continues to serve in leadership roles in the Manufacturing Research Center, the Model Based Systems Engineering Center, the Keck Virtual Factory Lab, and the Tennenbaum Institute for Enterprise Transformation. His area of interest is discrete event logistics systems, and his focus is on engineering methods for describing, analyzing, designing, and optimizing them. His email address is leon.mcginnis@gatech.edu.