Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

BEHAVIORAL DEVS METAMODELING

Hessam S. Sarjoughian
Abdurrahman Alshareef

Arizona Center for Integrative Modeling & Simulation
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
699 S. Mill Avenue
Tempe, AZ, 85281, USA

Yonglin Lei

Simulation Engineering Institute
School of Information Systems and Management
National University of Defense Technology
109 Yanwachi Rd
Changsha, Hunan, CHINA

ABSTRACT

A variety of metamodeling concepts, methods, and tools are available to today's modeling and simulation community. The Model Driven Architecture (MDA) framework enables modelers to develop platform independent models which can be transformed to platform-specific models. Considering model development according to the MDA framework, structural metamodeling is simpler as compared to behavioral metamodeling. In this paper, we shed light on and introduce *behavioral metamodeling* for atomic DEVS model. Behavior specification for an atomic DEVS model is examined from the standpoint of the MDA framework. A three-layer model abstraction consisting of metamodel, concrete model, and instance model is described from the vantage point of the DEVS formalism and the Eclipse Modeling Framework (EMF), a realization of MDA. A behavioral metamodel for atomic DEVS model is developed in EMF Ecore. This metamodel is introduced to complement the EMF-DEVS structural metamodeling. Some observations are discussed regarding behavioral metamodeling, model validation, and code generation.

1 INTRODUCTION

A variety of methods may be used to represent time-based dynamics of systems. The behavior of a system, for example, can be modeled using set-theory, UML diagrams, and pseudo code. Each kind of model serves certain purposes and must ultimately be mapped to programming code suitable for execution in one or possibly multiple target simulators. A mathematical model is useful for defining a system's structure and behavior independent of software design and simulation technologies. UML Class and Statecharts diagrams, among others, are useful for designing complex modeling and simulating engines which may or may not necessarily have mathematical grounding. Computer code can be developed and/or partially generated based on mathematical or certain kinds of software specifications. Each of these methods has its strengths and weaknesses and none is currently considered to contain all the necessary capabilities required for generating executable simulation code.

The atomic and coupled models in the DEVS formalism (Zeigler, Sarjoughian, and Au 1997) are "metamodels". From the standpoint of MDA, DEVS has an abstract syntax and an execution semantics which together define a modeling language for discrete event systems. The set-theoretic DEVS models are abstract mathematical artifacts. An atomic DEVS has its elements defined, for example, as sets, functions, and relations. These model elements individually and collectively satisfy certain general abstract properties and constraints. For example, a model can receive a finite number of input events within a finite period of time at arbitrary time instances, process these inputs with state changes within a time period, and generate a finite number of output events. It is the responsibility of the modeler to show that the developed atomic models for a given target simulator satisfy the properties and conform to the constraints defined for the DEVS atomic formal specification.

In the MDA framework, a concrete atomic DEVS model for a system component, relative to its metamodel, has specific structural (e.g., inputs and states with possible specific values) and behavioral elements (e.g., state transitions for specific source and target states with assigned times to next events). The metamodel is a language within which concrete models can be developed. Furthermore, a concrete model may also satisfy constraints such as state variable types and state transitions sanctioned for specific application domains. Full-fledge behavioral DEVS metamodeling can support automatic conformance of concrete models to their metamodels. This capability can significantly reduce the amount of manual effort required to show concrete models satisfy their metamodel properties and constraints.

From a tool's perspective, a simulator such as DEVS-Suite (ACIMS 2015) is designed as a collection of UML classifiers and relations that capture some aspects of the set-theoretic atomic and coupled parallel DEVS models. These models can also be collectively referred to as a DEVS UML "metamodel". The inputs, states, and outputs, and internal, external, output, and time advance functions of the model are defined abstractly; they by themselves are not executable. For example, the data structure for input is defined as a pair (port-name and input-variable) where port has a string type and input variable has an entity type. Similarly, the external transition function is defined as a method with specific arguments, but without any actual implementations for the state transitions and conditions under which they are to be performed. As in its mathematical counterpart, a concrete atomic model must have instances of the port-name and input-variable attributes belonging to the UML classes and interfaces. The realization of the formal DEVS models as UML specifications is advantageous. UML includes abstractions such as data typing, return types, and control structures that enrich the abstract atomic DEVS model specification. These models can be transformed to partial code for programming languages using professional tools dating back to the 1990s.

Simulators such as DEVS-Suite do not explicitly account for domain-specific modeling. A modeler can develop domain-specific models using object-oriented modeling principles and design patterns. The domain-neutral contracts embodied in the DEVS UML models can be enforced in an ad-hoc manner using low-level techniques such as checking for data type compatibility and expected values for concrete models that are implemented in some specific programming languages. These contracts cannot account for domain-specific knowledge; they must be extended. This approach becomes complicated and unwieldy as scale and complexity of the system to be simulated increase. Such resulting simulators lack rich capabilities to support and develop domain-specific metamodels and also are unable to validate basic model properties and constraints such as data typing and legitimate state transitions, for example. MDA-based modeling, however, can lend itself to develop and automatically validate behavior of any domain-specific DEVS concrete model against its metamodel and by extension the general-purpose atomic DEVS model.

Given the above discussions, we can make a few observations. When concrete atomic DEVS models are developed using programming languages, it is difficult to ensure they conform to their abstract model. A substantial amount of effort is required to concretize behavioral abstractions. Therefore, it is important for the meta and concrete atomic models to be systematically related to each other as proposed in the MDA framework. This is especially important given that the challenging part of developing models of complex systems is specifying their behaviors. Therefore, we need an atomic DEVS metamodel which can support behavioral modeling (e.g., receiving sanctioned input events and legitimate state transitions with timing). Toward this goal, we propose behavioral metamodeling for the general-purpose and domain-specific atomic models using the Eclipse Modeling Framework (Steinberg et al. 2008). Consistency between these models can be specified and enforced (referred to as validated) with automation. Concrete models can be generated from their domain-specific metamodels. Behaviors contained in these metamodels can significantly reduce the amount of effort to create concrete models and improve their quality using automated code generation.

2 BACKGROUND

In this work, our goal is to develop concepts that can enable building a framework capable of specifying meta-behavior for atomic DEVS models that can be used to create concrete atomic DEVS models. Toward this goal, we employ Model-Driven Engineering (MDE) and in particular, the MDA framework with its EMF realization. Although there are a variety of DEVS-based modeling and simulation tools, in this work we use the DEVS-suite simulator for developing the proposed behavioral DEVS metamodel.

2.1 MDA and Model Layers

The Model Driven Architecture (MDA) framework has been proposed for developing software systems (OMG 2003). Its main concept is a four-layer model abstraction hierarchy. A key abstraction concept in MDA is for a classifier and its instances to form a two-layer hierarchy. A classifier has an abstract specification that can have one or more instances. Classifiers can be said to be universal and instances can be said to be specific. Every classifier is at a higher level of abstraction in relation to its instance. Instances are related to one or more classifiers via conformance relationship. This implies having complementary models each of which having a certain role to play and collectively provide a disciplined roadmap for developing software systems. Each higher-level layer provides capabilities that are more abstract as compared to those provided by lower-level layers. Conversely, each layer is built using the elements provided in the layer above.

A realization of the MDA approach consists of Meta Object Facility (MOF), Unified Modeling Language (UML), User Model, and User Object modeling layers (OMG 2003). At the meta-metamodel (M3) layer, the MOF has an Ecore specification for defining metamodels in the OMG's family of MDA languages. Defined using the UML metamodel, the M3 layer supports computation-independent metadata management, metadata services, model management, tag capability, and reflective operations among others. The metamodel (M2) layer can have models that conform to the M3 layer. The M2 layer is directed at platform-independent modeling. These models can be domain-specific. The Ecore at the M2 layer can be used to define concrete models at the M1 layer. The M0 layer is used to define instances of models specified at the M1 layer. The M3, M2, M1, and M0 layers support incremental development of models for component-based systems. It is useful to note that the separation of concerns in MDA is important for developing software system tools including simulators.

2.2 DEVS Atomic Model

The set-theoretic specification of parallel atomic model $\langle X, S, Y, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta \rangle$ is domain-neutral. Its input and output are defined in terms of port names and variables. The variables can be arbitrarily complex. Atomic models are responsible for handling differences in the input and output variables. From software design, appropriate I/O type consistency is required. For any user-defined (and domain-specific) model, the internal, external, and confluent, time advance, and output functions can have arbitrary logic as long as they satisfy the abstract definitions provided in the mathematical atomic model specification. A restricted specification of parallel DEVS called Finite Deterministic DEVS (FD-DEVS) (Hwang and Zeigler 2009) has been developed. Events and states are defined to be finite sets and external and internal events are allowed to occur at time intervals restricted to rational numbers. No time interval between one event and the next can be infinitely small. This is achieved by abstracting time to be rational instead of real numbers. When states are simple, possible state transitions can be enumerated and unreachable states, identified. These restrictions can simplify model validation for the EMF-DEVS modeling described next.

2.3 EMF-DEVS Atomic Model

The EMF-DEVS (Sarjoughian and Markid 2012) is proposed as a metamodeling approach for the parallel DEVS formalism. The basic aim is to define and validate DEVS metamodels using the Eclipse EMF framework. The EMF validation infrastructure is used to define the elements of DEVS models with a set of constraints defined according to the DEVS formalism and the target DEVS-Suite simulator which is implemented in the Java programming language. Structures of atomic and coupled meta-DEVS models can be modeled and validated. The generic capabilities provided in the EMF M3 and M2 layers are extended to support concrete models for the DEVS-Suite simulator. The EMF-DEVS metamodel can support input, output, and state sets as well as external, internal, output, and time advance functions. These abstract functions $\langle \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta \rangle$ do not include the logic that is necessary to define behaviors. For example, the external transition function δ_{ext} does not define a generic transition from a source state to a target state with constraints and the output function λ does not define conditions for generating outputs.

In the context of metamodeling as in EMF-DEVS, the term validation refers to the Eclipse EMF validation framework and its execution engine. The Eclipse EMF has built-in validation mechanisms such as reflection for the metamodels at the M2 layer. Metamodels at the M2 layer can be validated for conformance to the meta-metamodel at the M3 layer. Concrete models at the M1 layer can also be validated to conform to DEVS metamodel. Here validation does not refer to execution of a metamodel over some period of time and determine whether or not it produces behavior per user requirements and expectation. Given a concrete simulation model (M1 layer), it can be verified to be specified correctly both in terms of M1 and M2 layers. When executed over some period of time and its behavior is recognized as acceptable for some defined experimental condition, the model is said to be valid. With respect to the verification and validation definitions for concrete models, the EMF-DEVS validation may be referred to as verification when a metamodel has domain knowledge (e.g., external transition function has the necessary control structure and other details to specify next state of a model given its current state and received input).

3 RELATED WORK

In this section, we primarily focus on behavioral DEVS atomic metamodeling and briefly consider the extent in which detailed specifications can be supported. Model-driven design approaches have been playing a greater role in developing complex simulation models. Focusing our attention on the OMG MDA framework and DEVS, we find some approaches that follow the MOF Technology Space (Bézivin and Kurtev 2005). In (Lei et al. 2009), a DEVS metamodel is devised for developing SMP2 (Simulation Model Portability standard). This metamodel is mapped to SMP2 metamodel using QVT (OMG 2003). Basic simple states and state transitions for atomic DEVS model are supported. In (Cetinkaya, Verbraeck, and Seck 2012), structural DEVS metamodeling can be supported. As in EMF-DEVS, behavior specification for atomic DEVS metamodel is not supported (see Section 2.3).

In the MOF technology space, some works have employed DEVS Natural Language (DNL), XML Schema, and Extended BNF for defining DEVS models. These support behavioral modeling using mostly the same ideas and methods. The MS4Me (Seo et al. 2013) focuses on modeling using DNL (Zeigler and Sarjoughian, 2012). The DNL as meta-language supports Finite-Deterministic DEVS models (Hwang and Zeigler 2009). MS4Me uses Xtext (Xtext 2013) to enforce DNL rules for simple inputs, outputs, states, state transitions, and timing. As a modern Java-like language, Xtend (Xtext 2013) supports developing FD-DEVS models. The MS4Me models can be augmented to become Parallel DEVS models using the full expressiveness of the Java language. It supports adding Java code to the model and thus developing Parallel DEVS models while maintaining a tight connection with the FD-DEVS models. The Java code is injected into slots in a structured manner using tagged code blocks. These are inserted directly into the generated source files. These tagged code blocks are used to specify additional behavior for initializing, internal transition, external transition, and output. Compared with FD-DEVS, classic or parallel DEVS models that have these kinds of code blocks are difficult to validate. The DEVSML (Mittal, Risco-Martín, and Zeigler 2007) is developed to for DEVS simulation models that can be executed in net-centric computing environments.

Some works employ SysML (Nikolaidou 2008) and UML (Borland 2003) (Risco-Martín et al. 2009) (Mooney and Sarjoughian 2009) (Pasqua et al. 2012). A SysML profile is developed for classical DEVS. An atomic model is defined as a collection of stereotype blocks. The behavior is defined by States Definition and Association diagrams. Atomic Internal and External diagrams are defined for the internal and external functions, respectively. The time advance and output functions are defined as part of the Atomic internal diagram. Similar to the above approaches, simple states with constraints are defined. The external diagram follows FSM with control elements such as choice, fork, and join elements. Time allocated to states can only be defined in the internal diagram. The DEVS SysML profile and DEVS MOF are intrinsically different due to their technology spaces. There exist other approaches that use "metamodeling" abstraction (Fard and Sarjoughian 2015; Ighoroje, Maïga, and Traoré 2012; de Lara and Vangheluwe 2004). A survey discusses uses of some MDE approaches for DEVS (Garredu et al. 2014).

4 ATOMIC DEVS METAMODELING

The mathematical properties and constraints defining an atomic DEVS model can be applied to any implementation of it. Therefore, it is useful to have a framework that can not only capture the atomic model's formal specification (i.e., a metamodel), but also enforce its syntax and semantics for domain-specific metamodels. Another important advantage is to define models independent of any particular simulator—i.e., metamodels can be transformed to concrete models that can be executed in simulators that are implemented in specific computing platforms. This framework must (help) validate *behavior* of any concrete atomic DEVS model against its metamodel. To achieve this, we propose introducing *behavioral metamodeling* to structural metamodeling. The resulting metamodeling framework must also lend itself to developing metamodels for modelers' domains of interests. This framework is also desired to support defining domain-specific concrete models for desired systems.

We intuitively define behavioral metamodeling as a set of concepts realized in a framework that supports specifying operational details of the internal, external, output, and time advance functions of any atomic DEVS model. These generic operations can be used to define behavior for any domain-specific DEVS metamodel. Domain-specific behavior can be specified by extending the generic DEVS metamodel behavior. That is, behavior of these functions are defined independent of computing platforms in which they can be fully implemented. The properties and constraints in the domain-neutral and domain-specific functions for the concrete models can be validated. The properties and constraints of the functions that are not satisfied in any concrete model are automatically identified and reported.

Figure 1 illustrates the concept of "meta" and "concrete" mathematical and UML modeling. The structure, unlike behavior, of mathematical atomic and coupled DEVS models can be completely specified both abstractly (as a metamodel) and concretely (as a concrete model). In mathematical modeling, a concrete model has more information relative to its metamodel. In the metamodel, $\delta_{ext}, \delta_{int}, \delta_{conf}, \lambda$ and ta functions are abstract mathematical constructs. The abstract atomic DEVS model functions do not have sufficient details, for example, as in Statecharts. Indeed Statecharts also does not capture the levels of detail in the functions that an arbitrary atomic model can have. In contrast, arbitrary concrete atomic models must have details including decision logics and control in the state, output, and timing functions.

The concept of meta and concrete models in UML are distinct as compared with the ones just described for a mathematical model. While UML metamodels are independent of computing platforms, concrete-models are not. Separating models to be platform independent and platform-specific is important (see Section 2.1). Meta models are technology (simulator) agnostic. Concrete models include details that are specific to target simulators. The meta and concrete models can be related to one another.

Focusing on behavioral modeling, the line arrows from the concrete model and metamodel are conceptual. For mathematical modeling, one may construct relationships to show, for example, state transitions in an external transition function in a concrete model conform to the abstract external transition function specification. In UML modeling, one can include rules that can be applied to concrete models. The block arrows at the metamodel and concrete model levels involve complex modeling and software development tasks, requiring detailed design and code development.

Considering the distinct roles mathematical and UML modeling offer, a desirable goal is to support both. The EMF framework (Steinberg et al. 2008) is a strong candidate as it already supports UML meta- and concrete modeling and it can support developing specific metamodels as in EMF-DEVS. In particular, the relationship between meta (M2 layer) and concrete models (M1 layer) is formalized. Furthermore, the EMF includes the metametamodel (M3 layer) and instance models (M0 layer). Given these, we extend the EMF-DEVS (Sarjoughian and Markid 2012) structural metamodeling to enable behavioral (functional) metamodeling. Generic and domain-specific metamodels with built-in and user-defined properties and constraints for the external, internal, output, and time advance functions are supported. Modelers may develop metamodels in a structured setting, thus leading to automation of metamodel validation as defined in EMF. (We note that validation is not referring to simulation validation.) Constraints defined for the generic and domain-specific atomic DEVS metamodels enable validating concrete atomic models.

4.1 Meta-behavior Modeling in EMF

We begin by sketching the basic details of the M2, M1, and M0 layers for the atomic DEVS model shown in Figure 1. At the M2 layer, the Ecore is an instance of the Ecore at the M3 layer. The M3 Ecore metamodel is at a higher level of abstraction with respect to the atomic DEVS metamodel. That is, the DEVS metamodel extends the instance of the M3 Ecore. The role of the M2 layer is to support developing concrete models at the M1 layer.

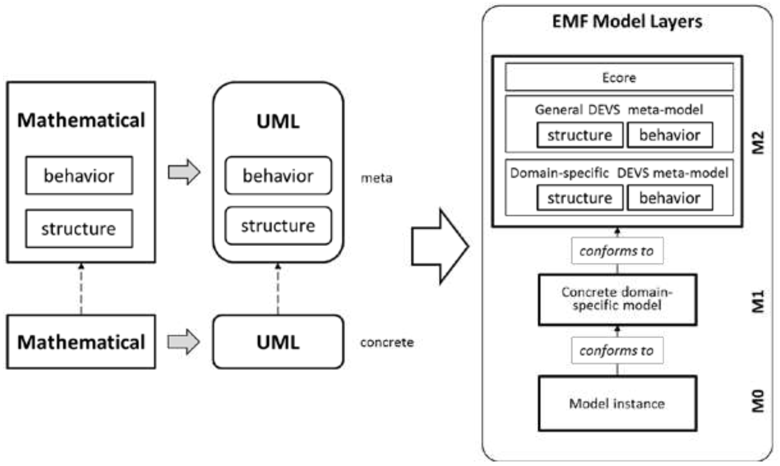


Figure 1: From mathematical to UML to EMF modeling.

As noted earlier, the DEVS-Suite simulator is developed in Java, a strongly typed language. The kernel of the modeling engine contains data structures and operations that satisfy the DEVS modeling formalism. Thus, at the M1 layer, user-defined models can be generated from the DEVS metamodels. Suppose we want a *Processor* model which can receive bags of input, process one of them, and generate one or more outputs. Assuming we have an *eProcessor* metamodel, it can be used to create the concrete Processor model. This concrete model at the M1 layer can be created for a platform-specific simulator such as DEVS-Suite. An instance of the concrete model at the M0 layer can be executed by the DEVS-Suite simulator.

In MDA, the M0 layer refers to the instances of the user models. These can be physical objects or executable software objects (e.g., compiled code). Such instances can be modeled as UML Object diagrams. As software objects, they can exist at execution time and their states may be stored, for example, as XML or byte code. In contrast, for simulation, the M0 layer refers to the user's parameterized atomic and coupled models. Therefore, at this layer, we have not only parameterized models but also their instances as part of other coupled model instances (see Figure 1).

Although metamodeling is not as expressive as programming languages such as Java, it is shown to be useful, for example, as in the Graphical Modeling Framework (Gronback 2009). The metamodel behavior specification for DEVS functions is achievable using Statecharts (Harel 1987). The elements of a parallel atomic model at M1 can be arbitrarily complex. An example is the external transition function. It can have any attribute type, expressions, and control structures that a target computing platform supports. The signature definitions for the atomic model external and internal transition functions can be defined using structural metamodel as in EMF-DEVS. The abstract definitions for these two functions must include some operations needed to result in some appropriate state change. State changes in these functions can be defined as transitions amongst source and target states. A transition may have input event, condition, and actions. A prototypical state transition is defined to transition from a source state to a target state. Such a constraint for state transitions can be defined and validated at the M2 layer. The output and time advance functions can also be defined using operations and control structures. An operation can have attributes and statements (McNeill 2008). A metamodel behavior specification requires identifying abstractions for state transitions in the external, internal, and confluent transition functions. Similarly appropriate abstractions are needed for the output and time advance functions at the M2 layer. The behavior of all DEVS functions as just described can be validated using EMF. The definitions for the atomic model functions must be consistent with the abstract DEVS simulation protocol.

In order to model the content of EOperation, we need to extend the EMF Ecore metamodel (McNeill 2008). Therefore, we will extend the Ecore metamodel to model DEVS functions that have been defined as EOperations (i.e., interface definitions) in EMF-DEVS. Our goal is not just to validate domain metamodels. We also aim to execute these functions after concrete models are generated for a specific simulator, DEVS-Suite for instance. The code generation creates the corresponding code for the defined elements in the metamodel. In EMF, the generator model plays a significant role in how the resulting code could be generated and organized via some settings that may differ based on the targeted platform. Those settings can be configured separately to ensure that the model maintains its platform independency. The process can be manipulated in a way that will lead to producing concrete models.

Thus, the general metamodel, shown in Figure 2, extends the EMF Ecore metamodel with some definitions for state transitions, actions, and conditions, basic elements of the atomic DEVS model. The metamodel extends Ecore elements with DEVS functions and also others for defining behavior. By extending Ecore, we are enabling EOperation (which is used to define DEVS functions) to include some content which can be transformed into concrete code rather than just having operation signatures. The extended EOperations will be contained in the extended EClass (eAtomic in our case) since they cannot be contained in EClass itself. This is a reason for extending EClass and EPackage since the Ecore elements themselves (EClass and EPackage) will not allow adding the extended ones (Extended EClass and EOperation) (McNeill 2008). Therefore, we first extend EOperation as a basic step to support behavioral DEVS metamodeling. Second, we extend EClass to allow adding the extended EOperation. The third step is extending EPackage to allow adding the extended EClass.

The second part of the metamodel (shown in the middle of Figure 2) is specializing eDEVSOperation to represent external transition, internal transition, output, and time advance functions. All of these can include operations that have statements and local variables. They also may have return values. The eDeltExt and the eDeltInt represent external transition and internal transition functions. Both compose transitions defined to capture the concept of state transition. State transition has a name defined as an EString, source and target defined as an ETypedElement, input defined as an optional reference of type eInput to be used in the external transition function. It can also have some actions and conditions. We also added two specialized state transitions for the phase and sigma primary states. Source and target phase are added to the state phase transition (StatePhaseTransition) and defined as an EString. Source and target states for sigma are added to the state sigma transition (StateSigmaTransition) and defined as an EDouble. Any other specific state transition can be also defined in the same manner for domain specific models. The behavior is consistently captured at the general and domain-specific metamodeling at the M2 layer. The generic behavioral metamodel is predefined for the modeler. The domain specific meta-behavior can be defined by the modeler as needed. The same approach is followed for the actions and conditions that are defined abstractly and then specialized to provide the support for developing the behavior at the concrete model. The eOutput and eTA elements refer to the eState in addition to the inherited composition feature from eDEVSOperation to support having other operations for more functionalities.

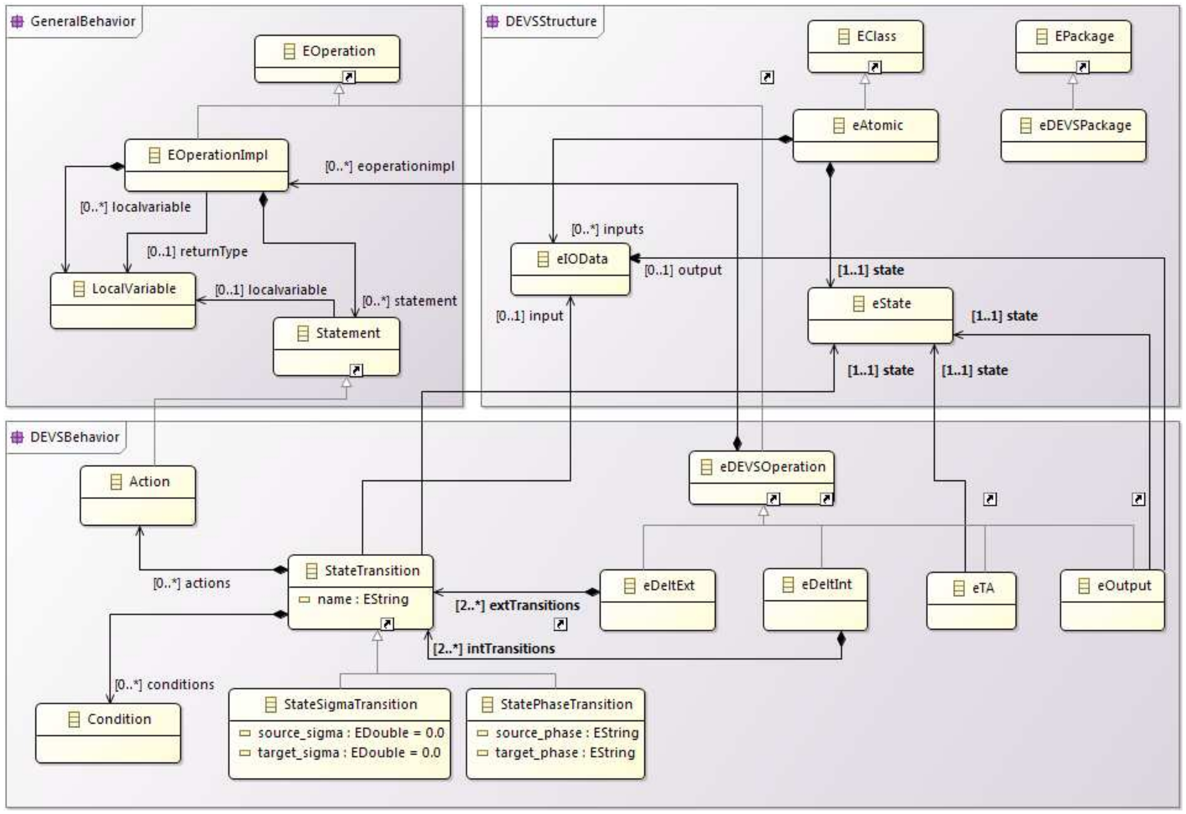


Figure 2: A metamodel for atomic DEVS Model with state transitions.

4.2 Constrained Meta-behavior Modeling

The metamodel shown in Figure 2 is based on the parallel atomic DEVS model. This model has an infinite state-space and therefore model validation (as in model checking) is impractical. A sub-class of DEVS called Finite-Deterministic DEVS (FD-DEVS) (Hwang and Zeigler 2009) has finite state-space which makes it attractive for behavior modeling at the M2 layer. The total state of the atomic DEVS metamodel can be defined as $\{primary\} \times \{secondary\} \times \mathbb{R}_{[0,\infty]}$. An atomic FD-DEVS model restricts the range of values for the time advance function to $\mathbb{Q}_{[0,\infty]}$. Model validation is computable when the values for inputs, outputs, and states (including time to next event) are finite. These constraints can be validated for having legitimate output, time advance, and internal and external transition functions. Constraints for state transitions (belonging to both external and internal transition functions) can be validated. For example, states in any state transition can be validated to include only the states defined in the model's state set and there are no unreachable states. For the external event, its input event can be checked to be included in the input set. State to output mappings can also be validated by checking whether or not every output belongs to the output set. We can also check if outputs are computed using states that belong to the state set. Time to next event for every state transition must also belong to $\mathbb{Q}_{[0,\infty]}$. When the time interval is infinity, three is no output. Validation of behavior domain-knowledge can be augmented with user-defined constraints.

Considering a domain-specific metamodel, they may have their own constraints on the input, output, and state sets as well as the atomic model functions. These constraints must be defined by the user, for example, by extending the EMF-DEVS metamodel. Users may specify domain-specific constraints using the EMF Eclipse framework and tool. Of course, user-defined constraints cannot contradict those that are defined for the generic metamodel. We note that the restrictions in the atomic FD-DEVS model and its dynamics may require complex control structures. State transitions in the external (or internal) transition function may have to be synthesized in complex patterns. Transitioning between external and internal transition functions can have many configurations. Similarly, the output and time advance functions may have complex structures. These considerations restrict the behavioral metamodeling describe above. Nonetheless, the capabilities afforded by MDA is advantageous as compared with model development where there is little or no means to start from metamodeling and reach executable models. Specific state transitions can be individually validated at the M2 layer. Behavioral metamodeling developed in this research aids model validation before transforming them to an M1 model and M0 simulation. Once concrete FD-DEVS models are generated from metamodels, they can be validated using existing techniques and tools (Dill 1990, Hwang, and Zeigler 2009)

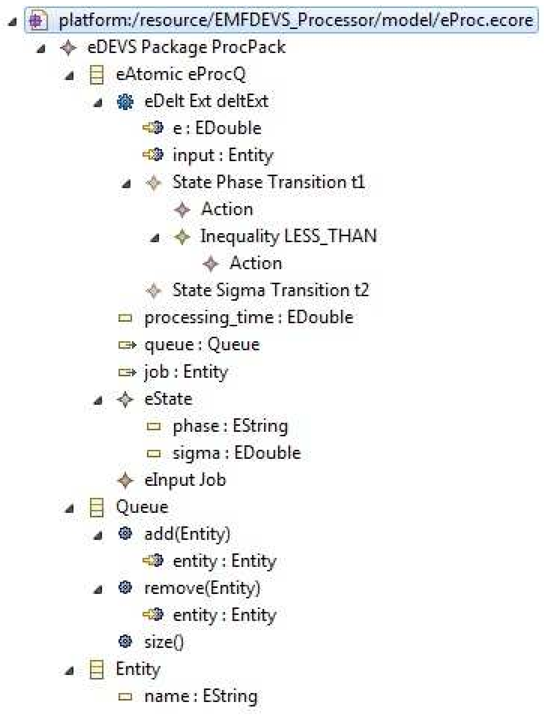
5 A PROCESSOR EXAMPLE BEHAVIORAL METAMODEL SNIPPET

In this section, we will demonstrate the process of developing a domain specific model (eProcQ as shown in Figure 3), which represents a simple processor with a queue. The processor metamodel is developed using the definition provided at the atomic DEVS metamodel. The root element is eDEVSPackage, which can contain the eAtomic models such as eProcQ and any other EClass such as Entity and Queue. Entity and Queue EClasses are defined similarly to their definition in the DEVS-Suite GenCol library (ACIMS 2015). Figure 3.a shows all the model elements in the EMF editor and Figure 3.b depicts the corresponding Class Diagram for the eProcQ Ecore model. Detailed specifications are provided for the external transition function relative to other modeled elements such as model states and variables.

We created two transitions and gave the values associated with each one. The first transition is for the phase and the other one is for the sigma. Figure 3.c shows the specified properties for the state phase transition that complies with the state phase transition definition. The phase transition has a condition and an action. The condition is modeled as an inequality for the queue size and the action is modeled as a method call for add operation, which is defined in Queue EClass. The action allows specifying the object, an action name that can be any operation associated with that object, and parameters. All of them have been defined as EReferences to their targeted model elements (see Figure 3.d). Figure 3.e shows an inequality condition specified based on the queue size. It has a left hand side which is specified as an action (queue.size() as shown in Figure 3.f) and right-hand side which is specified as an integer value of type EInt in this case. Currently, the metamodel is limited for only those scenarios since they are the only scenarios defined within the atomic DEVS metamodel. The implementation is done on a Windows 7 Computer. The models are created using Eclipse Mars Milestone 6 with Eclipse Modeling Tools and EMF Ecore 2.11.

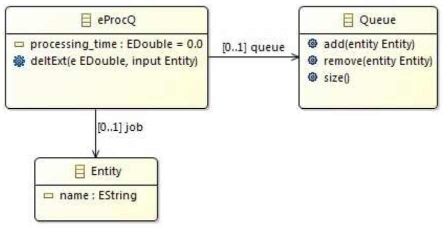
6 CONCLUSIONS

The term metamodel invokes different understandings since it refers to some model abstracted to another. It can encompass theories, methods, tools and domains of discourse including simulation. As such, “metamodeling” is used by theorists, developers, and practitioners in software and simulation engineering, among others. In this paper, we considered the modeling formalisms, and in particular asked at what levels of abstraction can the behavior of a prototypical atomic DEVS model be specified. Our inquiry is to distinguish meta-, concrete, and instance modeling layers from the standpoint of Model Driven Architecture. These layers can form a basis for building a new generation of modeling and simulation frameworks and tools that can help move from metamodeling to simulation code step-by-step. It is helpful to have modeling methods with tools that can not only represent mathematical abstractions within the MDA layers, but also introduce capabilities to enforce verification and validation as much as possible in the M2 before resorting to the M1 and M0 layers.

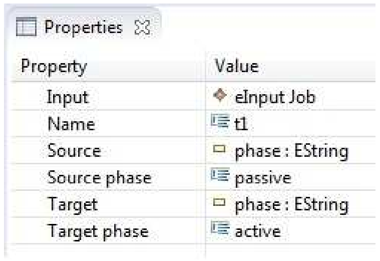


```
platform:/resource/EMFDEVS_Processor/model/eProc.ecore
  eDEVS Package ProcPack
    eAtomic eProcQ
      eDelt Ext deltExt
        e : EDouble
        input : Entity
        State Phase Transition t1
          Action
          Inequality LESS_THAN
            Action
        State Sigma Transition t2
      processing_time : EDouble
      queue : Queue
      job : Entity
      eState
        phase : EString
        sigma : EDouble
      eInput Job
    Queue
      add(Entity)
        entity : Entity
      remove(Entity)
        entity : Entity
      size()
    Entity
      name : EString
```

(a) Ecore editor view for the processor



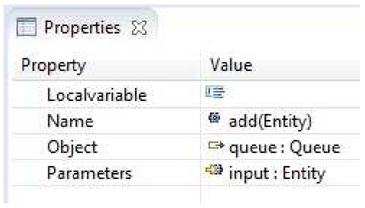
(b) A class diagram for the processor



Properties

Property	Value
Input	eInput Job
Name	t1
Source	phase : EString
Source phase	passive
Target	phase : EString
Target phase	active

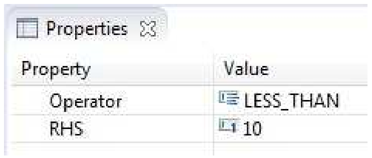
(c) Phase change for Transition t1



Properties

Property	Value
Localvariable	
Name	add(Entity)
Object	queue : Queue
Parameters	input : Entity

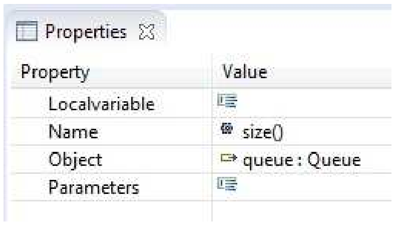
(d) Action for Transition t1



Properties

Property	Value
Operator	LESS_THAN
RHS	10

(e) Less than inequality for Transition t1



Properties

Property	Value
Localvariable	
Name	size()
Object	queue : Queue
Parameters	

(f) Left hand-side for the less than inequality for Transition t1

Figure 3: Ecore for a processor with primary state transitions for the external transition function.

One of the challenges facing building such ideal modeling and simulation tools is the difficulty of specifying behavior of models. We focused our attention on the atomic DEVS model. We proposed defining meta-behavior for general and domain-specific modeling using the concept of state transition from Statecharts for external and internal transition functions (see Figure 3). We then extended the EMF Ecore operation with the external, internal, output, and time advance functions. These functions, unlike the mathematical counterparts, can have some of their behaviors defined. These functions can also be validated to a limited degree. To validate, we described the necessity of restricting DEVS to Finite-Deterministic DEVS. We developed an example to show behavioral metamodeling for the atomic DEVS model. We focused this paper on the platform-independent metamodeling and briefly discussed its role for developing platform-specific tools.

Looking further into metamodeling, we observe that a target simulator must lend itself to the behavior defined in terms of state transitions, output, and time advance functions. Each function can have parts that are arbitrary and specific to the system being modeled. Thus, mapping behavior at a higher-level abstraction (as in the M2 layer) to lower-level abstractions (as in M1 and M0 layers) involves execution semantics (e.g., simulators may handle simultaneous event and communication differently despite being consistent with the abstract simulation protocol). Thus, it is desirable to lift behavior modeling as much as possible to the M2 layer with support to checking syntax and semantics with as little dependency as possible on the M1 and M0 layers, it is necessary to account for simulator design/implementation choices.

Knowing the high degree of DEVS expressiveness and the MDA framework, it is easy to see approaches that such as FD-DEVS should simplify development of verification and validation methods and tools. The degree to which the behavioral meta-model may be applicable to other kinds of modeling formalisms also remains as future work. In particular, for models that cannot be represented as DEVS, our approach for specifying meta-behavior may turn out to be useful. Finally, we believe exciting, challenging theoretical, methodological, developmental, and practical research remain to be formulated and answered for achieving general and domain-specific multi-layer behavioral modeling including meta-modeling.

REFERENCES

ACIMS. 2015. *DEVS-Suite Simulator.* Vers. 3.0.0. http://devs-suitesim.sourceforge.net/.

Bézivin, J., and I. Kurtev. 2005. "Model-based Technology Integration with the Technical Space Concept." *In Proceedings of the Metainformatics Symposium.* New York, NY.

Borland, S. 2003. *Transforming Statechart Models to DEVS.* McGill University.

Cetinkaya, D., A. Verbraeck, and M. Seck. 2012. "Model Transformation from BPMN to DEVS in the MDD4MS Framework." In *Proceedings of the Theory of Modeling and Simulation - DEVS Integrative M&S Symposium.* Orlando, FL.

de Lara, J., and H. Vangheluwe. 2004. "Defining Visual Notations and Their Manipulation Through Meta-Modelling and Graph Transformation." *Journal of Visual Languages and Computing* 15 (3-4): 309-330.

Dill, D. L. 1990. "Timing Assumptions and Verification of Finite-state Concurrent Systems." In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems.* 197-212. New York: Springer-Verlag.

Fard, M. D., and H. S. Sarjoughian. 2015. "Visual and Persistence Modeling for DEVS in CoSMoS." *In Proceedings of the Theory of Modeling and Simulation - DEVS Integrative M&S Symposium.* 962-969. Washington DC.

Garredu, S., E. Vittor, J. Santucci, and B. Poggi. 2014. "A Survey of Model-Driven Approaches Applied to DEVS - A Comparative Study of Metamodels and Transformations." *International Conference on Simulation and Modeling Methodologies, Technologies and Applications.* 179-187. Vienna, Autria.

Gronback, R. C. 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional.

Harel, D. 1987. "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming* (Elsevier) 8 (3): 231-274.

Hwang, M. H., and B. P. Zeigler. 2009. "Reachability Graph of Finite and Deterministic DEVS Networks." *IEEE Transactions on Automation Science and Engineering* 6 (3): 468-478.

Ighoroje, U., O. Maïga, and M. Traoré. 2012. "The DEVS-driven Modeling Language: Syntax and Semantics Definition by Meta-modeling and Graph Transformation." In *Proceedings of the Theory of Modeling and Simulation - DEVS Integrative M&S Symposium.* Orlando, FL.

Lei, Y., W. Wang, Q. Li, and Y. Zhu. 2009. "A Transformation Model from DEVS to SMP2 based on MDA." *Simulation Modelling Practice and Theory* (Elsevier) 17 (10): 1690-1709.

McNeill, K. 2008. "Metamodeling with EMF: Generating Concrete, Reusable Java Snippets." http://www.ibm.com/developerworks/library/os-eclipse-emfmetamodel/.

Mittal, S., J. L. Risco-Martín, and B. P. Zeigler. 2007. "DEVSML: Automating DEVS Execution Over SOA Towards Transparent Simulators." In *Proceedings of the DEVS Integrative M&S Symposium.* 287-295. Norfolk, VA.

Mooney, J., and H. S. Sarjoughian. 2009. "A Framework for Executable UML Models." *In Proceedings of the DEVS Integrative M&S Symposium.* San Diego, CA.

Nikolaidou, M., Dalakas, V., Mitsi, L., Kapos, G.D. 2008. "A SysML Profile for Classical DEVS Simulators." *The Third International Conference on Software Engineering Advances.* 445-450. Sliema, Malta.

OMG. 2003. *MDA Guide.* http://www.omg.org/cgi-bin/doc?omg/03-06-01.

Pasqua, R., D. Foures, V. Albert, and A. Nketsa. 2012. "From Sequence Diagrams UML 2.x to FD-DEVS by Model Transformation." In *Proceedings of the European Simulation and Modelling Conference.* 463-471. Essen, Germany.

Risco-Martín, J. L., J. M. Cruz, S. Mittal, and B. P. Zeigler. 2009. "eUDEVS: Executable UML with DEVS Theory of Modeling and Simulation." *Simulation Transactions* 85: 750-777.

Sarjoughian, H. S., and A. M. Markid. 2012. "EMF-DEVS Modeling." In *Proceedings of the Theory of Modeling and Simulation - DEVS Integrative M&S Symposium.* Orlando, FL.

Seo, C., B. P. Zeigler, R. Coop, and D. Kim. 2013. "DEVS Modeling and Simulation Methodology with MS4 Me Software Tool." In *Proceedings of the Theory of Modeling & Simulation - DEVS Integrative M&S Symposium.* San Diego, CA.

Steinberg, D., F. Budinsky, M. Paternostro, and E. Merks. 2008. *EMF: Eclipse Modeling Framework.* Pearson Education.

Xtext. 2013. *Xtext 2.4.* http://www.eclipse.org/Xtext/documentation/2.4.0/Documentation.pdf.

Zeigler, B. P., and H. S. Sarjoughian. 2012. *Guide to Modeling and Simulation of Systems of Systems.* Springer.

Zeigler, B. P., H. S. Sarjoughian, and V. Au. 1997. "Object-oriented DEVS." *AeroSense'97.* 100-111. Orlando, FL.

AUTHOR BIOGRAPHIES

HESSAM S. SARJOUGHIAN is an Associate Professor in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University (ASU), Tempe, AZ, and co-director of the Arizona Center for Integrative Modeling & Simulation (ACIMS). His research interests include model theory, poly-formalism modeling, collaborative modeling, simulation-based science, and simulation tools. He is the director on the ASU Online Masters of Engineering in Modeling & Simulation. He can be contacted at <sarjoughian@asu.edu>.

ABDURRAHMAN ALSHAREEF is a Computer Science PhD student in the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University (ASU), Tempe, AZ, USA. He can be contacted at <alshareef@asu.edu>.

YONGLIN LEI is an Associate Professor of Simulation Engineering Institute at the National University of Defense Technology, Changsha, China. His research interests include model composability, model driven architecture, domain specific modeling, and their applications in defense simulations. He can be contacted at <yllei@nudt.edu.cn>.