DEVS SPECIFICATION FOR MODELING AND SIMULATION OF THE UML ACTIVITIES

Abdurrahman Alshareef
Hessam S. Sarjoughian
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
alshareef@asu.edu, sarjoughian@asu.edu

ABSTRACT

Recently, the existence of the foundational UML subset (fUML) has made a significant impact on driving
the mainstream in behavioral modeling. Recent advances on concepts like UML actions and behavioral
metamodelling promise better mechanisms in handling behavior. We argue that a greater potentiality can be
achieved through establishing a rigorous mathematical grounding for the subset. We also argue that Parallel
DEVS can be used as a suitable candidate. Thus, we propose formalizing the creation of UML activities as a
set of atomic and coupled DEVS models. These abstractions have distinct structures and behaviors. That is
to say, the foundational elements of UML activities and actions are modeled and mapped into a set of atomic
as well as coupled models. Then, they can collectively serve as a basis for grounding different diagrams via
coupling. We demonstrate the use of the formalism for the modeling and simulation of activities.

Keywords: Behavioral Modeling, DEVS-Suite, Executable UML, Parallel DEVS, UML Activities.

1 INTRODUCTION

The path toward utilizing the capabilities available in discrete event system modeling is extremely important.
We are currently still far away from fully enhancing the use and therefore preventing failures that may arise
in such systems. The concern of advancing and investigating discrete event systems has been continuously
growing in order to attain people appreciation of the potential solutions and consequently taking advantage
of them. UML has been dominantly used in the world of software modeling. Some aspects of it bear
resemblance with discrete event systems which have been subject to research in the last decade from two
points of view. The first one is to use UML as a language for system specifications while the other one is to
supply UML with concepts to overcome its weakness in terms of formal grounding. The potential value that
can be added to UML by providing a rigorous mathematical specification is priceless. It basically brings the
value of formal specification into a widely used language by modelers.

However, there is a substantial complexity that can be encountered during the path of bringing formal spec-
fication to such a language. First, we are dealing with a high level of ambiguity that is in some cases
intentional according to the purpose of the language or to maintain some level of generality. The language
has been standardized around the concept of allowing end users to easily comprehend its models in a human
understandable manner. Adding formal specification may potentially result in relatively sacrificing some
general concepts and definitions. This certainly makes the problem challenging in addition to the inherent complexity and ambiguity that might be encountered within the language itself.

The problem has been widely discussed in research from both theoretical and practical aspects. The solutions also widely vary. Researchers have been continuously providing extensions, frameworks, and tools. There are some solutions that rely on further knowledge about the intended domain while some others attempt to remain more neutral and domain independent. The same perspective has been also applied regarding with platforms and applications. The notion of behavioral DEVS metamodeling is introduced in a previous work (Sarjoughian, Alshareef, and Lei 2015) to provide some support for behavioral specification at the meta-levels. SysML is one of the most common UML profiles that has been devised based on similar motivation for system engineering and yet it is challenging to simulate (Nikolaidou et al. 2016).

Some current solutions do not incur the existing lack of formality. Others require a substantial amount of additions and extensions in order to resolve some ambiguities. On the other hand, many approaches take a different direction and address the problem from an implementation standpoint, for example, by operational semantics for well-formed models. The Model-Driven Architecture (MDA) as well has been extensively used to address the problem from an architectural point of view.

A suitable candidate to be used for activities is Parallel DEVS (Chow 1996). In so doing, the basic activity behavioral elements of UML (OMG 2012), which are mainly action nodes in activities, can be mapped to Parallel DEVS in the sense of their resemblance with atomic models. Elements of this subset may have inputs and outputs. The transition function can be then devised for each element to implement the behavioral specification according to their semantics. For example, the semantics of fork node can be specified. This realization can result in a conveniently visual representation and simulation of surface UML models that have a collective behavior in terms of its foundational elements such as in fUML (OMG 2016). This is achieved by utilizing a Parallel DEVS simulator such as DEVS-Suite (ACIMS 2015).

In this work, we begin by giving some background about related matters of UML activities and the foundational UML subset (fUML). We then discuss the related works. Next, we describe the basis for our rationale about the concept of activity modeling and simulation and what does it mean in terms of DEVS modeling. Then, we establish a fundamental ground for mapping between activities and DEVS concepts. We demonstrate the approach with an illustrative simple example. Finally, we discuss some findings and steps toward continuing the work in both near and long term future.

2 BACKGROUND ABOUT FOUNDATIONAL UML SUBSET

Our work has been conducted after an extensive research on the state of the art of UML standards and the recent advancement of the fUML and concepts like executable modeling. Therefore, it would be useful to provide a basic but yet sufficient background of the so-called fUML and its related subjects.

2.1 UML Activities

The fUML subset is devised based on the approach of activity modeling. From a high level, an activity can be seen as a directed graph of vertices and connected by edges. Each vertex is an activity node and each edge is an activity edge which can be either an object or a control flow. Every control, object, or executable node is an activity node. Action is an executable node that can be further specialized to encompass a variety of different basic behaviors. Previously, we discussed in details the use of activities in specifying the behavior of atomic DEVS models (Alshareef, Sarjoughian, and Zarrin 2016). Lifting behavior to higher levels across the meta-layers such as M2 can result in greater benefits especially with respect to checking the syntax and semantics of the specified models at a higher level of abstractions. As a result, modeling at the lower levels
becomes simpler and less detailed since the other details about the system under development have been addressed at the higher levels. In addition, we extend our research toward a direction on simulating UML activities by exploiting the capability provided in Parallel DEVS simulator.

Parallel DEVS was presented in (Chow 1996) to provide the capability of handling collisions that may arise during the interaction between different components. The formalism allows all imminent messages to be sent out simultaneously which can be used as a good abstraction for handling activities flows. This is especially important in the case where the activity node has multiple incoming flows such as the case in fork and merge nodes.

2.2 Abstract Syntax

The abstract syntax of fUML mainly consists of classes, common behaviors, activities and actions. It selects some certain elements from the complete UML in order to provide a more precise definition of their semantics. Although it uses the above four packages, it excludes some features thereof. Some packages from the UML 2 Superstructure are excluded. The reasons for excluding some packages or features vary. Some packages are excluded due to their insignificance in terms of execution. Others are excluded due to the encountered complexity if they are to be realized in a computational platform. In some occasions, features are excluded because of their generality and therefore some ambiguity or restrictions with respect to their semantics may arise.

The packages for activities and actions, which include mainly our proposed modeling elements, are included although some of their features are yet excluded. Overall, fUML does not entirely realize the complete model of activities as defined in the UML, but rather realize what is possible to be realized at a lower level. The included pieces of these packages are organized and sub-organized in a well-established architecture. The relationships are then established as necessary using dependency, specialization, and import. Actions are the fundamental units of behavioral specification. However, they have to be contained in some behavior which is currently activities. According to UML 2.5, action is a subtype of executable node which is itself a subtype of activity node. Therefore, actions can be contained in activities as executable nodes or interactions as actions. Currently, we only focus on their containment within activities. Action may have input pins, output pins, or constraints. Action is also specialized further to represent many different form of behavior based on their semantics and use within a model. For example, structural feature actions are used to handle reading, writing, adding or removing structural features such as a queue within some model. The manipulation of the feature has to go through this set of actions. There are multiple sets within the metamodel of actions. Each set has also more subsets or specific actions. The relationship between actions are determined mainly via generalizations from the more general or abstract action to the more specific or concrete ones. In our work, we handle the concept of action in general. However, we also go into details for some of the defined actions that are necessary for the discussion and demonstration. A similar approach then can be applied to other set of actions.

2.3 Execution Model

This is a major contribution of the fUML given that the abstract syntax is already defined in the UML superstructure and yet is being revisited in the subset. The execution model is an fUML model that defines the specification for the execution. The execution model expects a well-formed model in order to provide a meaningful execution semantics. The operational specification is currently described in the form of equivalent code in Java. The reason is because activity diagrams become inconveniently large when addressing relatively significant behavior. One of our concerns in this work is to overcome this issue by utilizing DEVS capabilities for addressing behavioral specification.
2.4 Foundational Model Library and Base Semantics

It can be also useful to provide a brief description and reasoning about these two components of the fUML specifications. The foundational model library includes primitive types and behaviors accompanied with their basic operations such as functions for handling Boolean signatures. It also defines some capabilities for handling input and output. These definitions along with other concepts are rigorously defined in the DEVS formalism as well as the modeling and simulation packages provided in the DEVS-Suite simulator (ACIMS 2015). Therefore, we focus our use on the formalism and what is provided in the selected simulator. Regarding the base operational semantics of fUML and Parallel DEVS, the simulator provides an explicit protocol for the behavior of the execution model which can be used for verification purposes. In our case, the semantics are substantially extended due to the definition of time. However, the base semantics is still maintained.

3 RELATED WORK

A significant effort has been continuing to enhance the process of model-driven practices for modeling in general and for simulation modeling as well. The notion of allowing models to be executable has been existing for a while. In (Harel and Gery 1996), an integrated set of languages are developed for object modeling around statecharts. The goal is to produce an executable model which cannot be achieved without defining a precise semantics. That is followed by an attempt to define formal operational semantics for UML statecharts (Latella, Majzik, and Massink 1999). Another executable UML has been introduced by (Mellor, Balcer, and Foreword By-Jacoboson 2002) to complement UML with code in order to make it executable using model compilers. The notion of the introduced executable model is inspired by approaches such as (Stahl, Voelter, and Czarnecki 2006). In (Kirshin, Dotan, and Hartman 2006), a UML simulator is defined based on a generic model execution engine. The simulator relies on the available knowledge in the model upon the start of the simulation and suspends when there is missing information via user/tool interaction. More recently, the fUML (OMG 2016) is proposed to provide the semantics necessary for executing a subset of UML. The fUML is used then in (Mayerhofer 2012) to enable model testing and debugging. These capabilities are provided in a model execution platform called MOKA within the Papyrus Eclipse (Eclipse Foundation 2016).

There has been also an effort to employ model-based and model-driven methodologies for the system-theoretic specification (Risco-Martín et al. 2009, Mittal and Martín 2013). In (Mooney and Sarjoughian 2009), DEVS is utilized for the creation of executable UML models based on statecharts. This approach, unlike other DEV5-based approaches, is grounded by providing both specification syntax and execution semantics with well-defined timing which are needed for concurrent handling of events for developing composite executable UML models.

Activities has been considered as a major modeling approach to resolve some major limitations in the current modeling practices. We characterize these efforts based on their purpose. Some efforts aim toward automating and producing models that are suitable for production, where the others are built based on the theory of modeling and simulation. The model is a foundational element from both perspectives. The latter can be also considered as a theoretical basis for general system design instead of just being tied to simulation purposes. Thus, on one end, this is an attempt to devise a methodology based on the DEVS formalism for activity modeling. On the other end, it provides a profound means for specifying their precise semantics.

Our methodology relies on a different perspective in approaching model execution. We consider that the creation of the activity model is conducted for simulation purposes despite of the fact that it is intended and often used for actual software product development. In fact, this is a useful practice. We want to ensure that the models are established based on rich system-theoretic specification while providing the same capabilities
that can be provided by debugging techniques is maintained such as visualizing, controlling, and tracing the execution yet in a disciplined manner. We take a similar position with (Risco-Martín et al. 2009, Mooney and Sarjoughian 2009), however, as in our previous work (Alshareef, Sarjoughian, and Zarrin 2016), we keep our focus on the activity modeling and try to achieve our new target goal exploiting the DEVS formalism.

4 ACTIVITIES SIMULATION THROUGH DEVS: FINDING RIGOR

There is some level of difficulty when it comes to modeling for simulation. A profound simulation for activities accounts for further knowledge and details beyond basic debugging capabilities provided in some approaches and tools. Although there are some temporal aspects in the process of model debugging (Mayrhofer 2012), the notion of time is not explicitly defined in the debugging modes. The definition of step is intrinsically defined in simulation modeling of dynamical systems based on time notion. In DEVS, time period assigned to any state change due to external and internal transition functions has arbitrary accuracy. Furthermore, the construct called elapsed time allows inputs to be handled by the external transition function at any future arbitrary time instance. These definitions can be effectively utilized to provide a stronger foundation for the simulation of activity modeling. Instead of using a step-wise or breakpoints as mechanisms to handle the execution of the activity model, the time advance function and the notion of sigma are used. The step can be then performed during runtime based on these definitions. Figure 1 shows an overall view of how the concepts of modeling and simulation are defined and employed in many current practices. We will make use of these entities as defined in (Zeigler, Praehofer, and Kim 2000) in order to better perform the task.

![Figure 1: A simplified view of employing concepts in M&S for activities modeling.](image)

We construct the activity simulation based on the hierarchical and modular simulation framework for DEVS simulator. The basic activity constructs are generally specified by a set of atomic models. Consequently, each construct can be then simulated. The coupling is defined between atomic models. The activity model is collectively defined via coupled models. The communication between elements is handled through messaging to represent the control as well as the object flow. The locus is transmitted to other components according to the semantics of the activity.

4.1 A DEVS Grounding for UML Activities

This subsection presents how UML activities are treated from DEVS standpoint, including their structural and behavioral properties. A mapping is proposed to facilitate the process of understanding the bridging points between activities and the DEVS formalism. The mapping includes the general constructs and more concrete constructs thereof. We also discuss the modularity and the generality of the mapping in subsequent sections. We shall begin with the activity nodes and then discuss the edges.

The activity node, which is the most abstract node element in activities, is generally specified by an atomic model. Most of the specialized activity nodes bear resemblance in terms of their structural properties as opposed to their behavior. Therefore, the specification of the atomic model needs to be specialized further with respect to the atomic model behavior in order to define the semantics of the corresponding activity node. The action, for instance, is treated as an atomic model with some input and output ports (see Figure 2). The behavior of the atomic model is defined by (external, internal, and confluent) transition, output, and
time advance functions. It is defined at the most concrete element in the current activity hierarchy. The activity node is specialized as control, object, and executable nodes among others. These elements are also specialized further. For example, the decision node is a subtype of the control node. Its semantics is defined in the behavior of the atomic model that corresponds to the decision node although its structural properties are defined at a higher level since all elements share the same structural characteristics. The expansion region is a special case of the activity node where coupled model is used for it. The reason is quite straightforward since the expansion region may contain multiple nodes and edges which in turn are mapped to their corresponding atomic models. The elements of any activity are unique and may also have unique relationships to other elements in the activity. This abstraction exactly mirrors that of flat DEVS coupled model specification.

The internal coupling is used to specify activity edge, which is a supertype for the control as well as the object (data) flows. The input and output ports make modeling easier and therefore used to specify the input and output pins alongside with ports to handle the flow. The activity node has at least one input port which is used to enable it. Note that, Parallel DEVS is selected since it allows for receiving input (event) values simultaneously via multiple input ports. The nodes vary in the number of their input ports based on their concrete types and incoming edges. This is, in a sense, similar to the activity nodes in having input or value pins, or not having any. The output ports are treated in similar manner except they account for the outgoing edges and output pins. We note that at a minimum, a non-trivial atomic model must have at least either external or internal transition state transition, two outputs, and two state variables one representing assignment of time duration for operations and another representing at least two values for the model to be in (Wymore 1993).

Thus, the previous components can serve collectively to form an activity model. The model is viewed as a coupled model from a DEVS perspective. The connection between different activity constructs is established by edges. In the DEVS formalism, the specification of the external input and output interfaces, components, and the coupling relation are included to serve as a means for establishing models from yet other DEVS models (Zeigler, Prahofer, and Kim 2000). The external input coupling specify the connection from the input parameter in the activity and consider it as an external input. This conforms to having two distinct components as required by the Parallel DEVS coupling legitimacy property. The coupling is connected with some component port. The external output port is also used in the same manner but for the activity output. The internal couplings are used as discussed in the previous paragraph to specify edges. Table 1 shows a subset of the mapping although additional elements might be needed to put the activity in a purely modular object-oriented modeling context. For example, internal couplings can be added for communication between objects. This communication can be requested by some object node in the activity model and then outputs are sent out to other actions accordingly. Other couplings are used for controlling the model with respect to the activity semantics. In the UML activity modeling it is not necessary to require for two components to strictly communicate via ports; i.e., a component can invoke operations of some other component instead of using signals (messages).
Table 1: A subset of the mapping for activity elements.

<table>
<thead>
<tr>
<th>Activity DEVS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity Coupled model</td>
</tr>
<tr>
<td>Activity node Atomic model</td>
</tr>
<tr>
<td>Expansion region Coupled model</td>
</tr>
<tr>
<td>Input and value pin Input port</td>
</tr>
<tr>
<td>Output pin Output port</td>
</tr>
<tr>
<td>Activity edge Internal coupling</td>
</tr>
<tr>
<td>Activity parameter External input coupling</td>
</tr>
<tr>
<td>Activity parameter External output coupling</td>
</tr>
</tbody>
</table>

4.2 The Semantics of Activities

The basis of any simulation environment for activities has to account for execution semantics. The objective is to define the semantics that is specific with respect to DEVS modeling and yet general in the context of activity modeling. The activity is initialized by either an initial node or some external influence. For example, the activity can be created in the context of some other model such as a class. Any valid flow can represent an execution of a particular activity performed by a course of actions. The control nodes manage that flow however they do not impose changes on the associated objects. Each control node, as well as object and action nodes, has its own semantics. Our objective is to define their semantics formally in a set of DEVS models. For instance, the semantics of a decision node can be defined in the behavior of its corresponding atomic model. Upon inputs arrival, the phase is changed to **executing** to denote the existence of an active node in this particular execution path. Then, in the next time step, the condition associated with this particular decision node is evaluated. The result of the evaluation will determine via which port the output sent out. If the evaluation of the guard condition is true for more than one case, the output will be sent out via one of the output ports arbitrarily. It is, however, the modeler responsibility to account for such a scenario if a mutual exclusion is required for instance. Finally, the phase is set back to passive by the internal transition function. The formal specification of the atomic model that corresponds to that is defined in Parallel DEVS as

\[
\text{DEV}_\text{S}_{\text{processing time}} = (X_M, Y_M, S, \delta_{\text{ext}}, \delta_{\text{int}}, \delta_{\text{con}}, \lambda, ta)\]

where

\[
I\text{Ports} = \{\text{"in"}, \text{"in1"}\}, \quad X_p = V \text{ (an arbitrary set)}
\]

\[
X_M = \{(p,v) | p \in I\text{Ports}, v \in X_p\} \text{ is the set of input ports and values}
\]

\[
S = \text{phase} \times \sigma \times \text{condition} \times \text{store}, \quad \text{where}
\]

\[
\text{phase} = \{\text{"passive"}, \text{"executing"}\}, \quad \sigma = \mathbb{R}^{+}_{0,\infty}, \quad \text{condition} = \{\text{true}, \text{false}\}, \quad \text{store} \in X_M
\]

\[
O\text{Ports} = \{\text{"out"}, \text{"out1"}\}, \quad Y_p = V \text{ (an arbitrary set)}
\]

\[
Y_M = \{(p,v) | p \in O\text{Ports}, v \in Y_p\} \text{ is the set of output ports and values}
\]

\[
\delta_{\text{int}}(\text{phase}, \sigma, \text{condition}, \text{store}) = (\text{"passive"}, \infty, \text{condition}, x) \text{ where } x \in X_M
\]

\[
\delta_{\text{ext}}((\text{phase}, \sigma, \text{condition}, \text{store}), e, X_M) =
\]

\[
(\text{"executing"}, \text{processingTime}, 1, \text{condition}, (p_1, v_1), \ldots, (p_n, v_n)) \text{ if } p_i \in \{\text{in}, \text{in1}\}, i \in \{0, \ldots, n\}
\]

\[
\delta_{\text{con}}(s, ta(s), x) = \delta_{\text{ext}}(\delta_{\text{int}}(s), 0, x)
\]

\[
\lambda(\text{"executing"}, \sigma, \text{condition}, \text{store})
\]
= (out, store.v) if condition = true and store.p = in,
= (out1, store.v) if condition = true and store.p = in1,
= (out, store.v) if condition = false and store.p = in1,
= (out1, store.v) if condition = false and store.p = in, where (p, v) ∈ X_M

The processing time is defined as an abstraction to represent the stepwise execution of the activity model. The other control nodes are defined in similar manner while distinguishing between their unique structural and behavioral properties. The fork and join nodes are used for branching in and out the flow with a synchronizing capability, that is, the join node waits for all incoming flow loci in order to transition to an executing state and the fork sends out loci via all its outgoing flows. The initial node does not have incoming flow and therefore no input ports are used for this purpose. Similarly, the final node does not have an outgoing flow. It should be noted that any output from an atomic or coupled model is automatically duplicated per number of couplings it has to input and output ports of other components according to the external input and output couplings as well as internal couplings.

5 NETWORK SWITCH: AN EXAMPLE

Grounding activity models into DEVS has been accomplished at the meta-layers. Thus, the process of creating concrete models becomes easier. We choose the network switch for several reasons. Obviously, it can be specified as a coupled model. It illustrates the handling of inputs in Parallel DEVS. It also exemplifies inherently different behaviors since it contains a switch as well as a processor. The model is described in (Zeigler, Praehofer, and Kim 2000) as shown in Figure 3. The internal transition function is applied before the external one in the case of having both the switch and one processor imminent. The switch decides to send out the job via one of its output ports based on its polarity. It either sends the incoming input from the first input port on the the first output port and similarly for the second input and output ports, or it reverses that if it is on the other setting. The nature of the switch in its general form resembles the semantics defined for the decision node of activity. From another perspective, the decision node can be considered as a higher level of abstraction of the switch. Therefore, the activity model in Figure 4 can be seen as a higher level abstraction of the coupled model.

![Figure 3: The network switch Parallel DEVS coupled model.](image)

In addition to the abstraction, the semantics of activities are incorporated as well. The behavior of the switch accounts for the semantics of the decision node. In another word, the polarity and inputs are checked as conditions. Once evaluated, the node will send out the job to a corresponding processor. Actions are treated as atomic models in general. However, they are treated as processor in this example and because of that the behavior of the processor accounts for the semantics of the action. That is to say, the semantics
of activities are specified in the set of atomic models that are then underpinned by the semantics of the simulation protocol for the DEVS formalism.

5.1 Modularity

Thanks to the closure under coupling property, we can ensure the feasibility of constructing a hierarchical model based on the elemental constructs. The mapping is established based on the most abstract activity constructs and then the behavior is specialized accordingly. The most concrete elements are mapped into atomic models and used in an activity or expansion region via coupled system specification. The coupled DEVS specification for the network switch activity is

$$ A = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC), $$

where

- $InPorts = \{“in”, “in1”\}$, where $X_p = V$ (an arbitrary set), $X_M = \{(v) | v \in V\}$
- $OutPorts = \{“out”\}$, where $X_out = V$, $Y_M = \{(“out”, v) | v \in V\}$
- $D = \{DecisionNode0, Action0, Action1\}$
- $M_{DecisionNode0} = DecisionNode; M_{Action0} = M_{Action1} = Action$
- $EIC = \{((Activity0, “in”), (DecisionNode0, “in”)), ((Activity0, “in1”), (DecisionNode0, “in1”))\}$
- $EOC = \{((Activity0, “out”), (Activity0, “out”)), ((Action1, “out”), (Activity0, “out”))\}$
- $IC = \{((DecisionNode0, “out”), (Action0, “in”)), ((DecisionNode0, “out1”), (Action1, “in”))\}$

Since this is also a system specification itself, it can be used further in a larger context within other system specifications. This also stands to provide additional benefits in the context of UML; however more investigation on the mapping has to be carried out given the current system specification that corresponds to only activities and their substances only. Various behavioral and structural subsets of the UML metamodel need to be investigated in order to determine how they can be treated in this larger context.

5.2 The Generality of the Models

From the same viewpoint that considers the simulation protocol to be domain-agnostic, we consider the DEVS models built for the execution semantics of activities to be also domain-agnostic relative to their activity model instances. The behavioral specification is polymorphic in the sense that they capture behavior that accounts for multiple types and values. The notion of specifying structure at the meta-levels is well...
Alshareef and Sarjoughian

established but not for the state transition behaviors (Sarjoughian, Alshareef, and Lei 2015). We think of
the created DEVS models along with their behavior to be situated at the M2 layer in the Model-Driven
Architecture. Their generic behavior can be used to simulate any specific instance. Despite that they are
extensible, their current behavioral specification is sufficient to be applied on well formed instances at the
concrete level M1. A view of the simulation for some of the high level activity constructs is shown in
Figure 5.

Figure 5: A simulation view for the high level activity constructs used to model network switch (implement-
ated in DEVS-Suite).

Another important aspect of the models is regarding metamodeling, that is, the behavior of these models can
be viewed as activity models. One activity corresponds to each function of the atomic model (Alshareef,
Sarjoughian, and Zarrin 2016). This is due to the existence of the meta-layers and the conformance relation-
ship between them. The models that encompass the semantics of the activities can be viewed as activities.
However, they are currently realized in Java code snippets that can be understood in the DEVS-Suite sim-
ulator. The representation of these snippets in activity notation can be thought of based on the Annex A
provided within fUML specification (OMG 2016).

6 FUTURE WORK

We are currently working on the development of two packages for DEVS-Suite for full support for the
simulation of any well-formed activity model. One package is to encompass the semantics of the different
control nodes as well as the action in its generality. The second package is for interpreting activities as a
prior step before simulating them.

The activity package contains the activity metamodel as discussed. It also contains generic code snippets
for the semantics of each control node. The action is treated as a general construct. Additional Research is
needed in order to support further elaboration for specific action types. This includes the concrete details of
the action type as well as a mechanism to incorporate that together within the current context.

The interpreter is also added to make it easier to incorporate activity model from the activity package stand-
point. A certain checking has to take place in order to ensure the injected models are well-formed. We can
benefit from that by eliminating as much of the code portions possible for target execution platforms. The
transformation to an executable form is restrictive, particularly from the standpoint of automatically gener-
ating code for behavior specified in atomic models (Sarjoughian and Elamvazhuthi 2009, Sarjoughian and
Markid 2012, Seo et al. 2013, Mittal and Martín 2013). Thus, the potential of the interpreter in automating
as much of the process while having a control over the abstraction levels can be promising. We are working
on this issue and our target objective is to ultimately provide these capabilities and make them accessible to
modelers who use simulators such as DEVS-Suite. The direction of the transformation in a general sense remains open for future research.

7 CONCLUSION

In summary, we proposed DEVS specification for UML activities in conjunction with the concept of executable modeling. The objective is to obtain a rigorous grounding for the modeling and simulation of activities based on system theory. We proposed a mapping for activity elements into DEVS models. The mapping is centered around the concept of activity nodes, including actions, since they collectively serve as a basis for the activity and connecting them via edges. In addition, we complemented that with the definition of their semantics. We demonstrated the approach by developing some example atomic and coupled models for a network switch according to the Parallel DEVS formalism with some remarks regarding the modularity and expressibility of the models.

The research on the behavioral modeling remains quite challenging. We employ a variety of concepts, frameworks, methodologies and tools in a manner that is consistent with formal model specifications. This is important in order to be sufficiently powerful to account for non-trivial dynamical systems and their models. Approaching the creation of executable modeling from a modeling and simulation standpoint is useful. The value of enabling rich and mature concepts such as experimental frame (Zeigler, Praehofer, and Kim 2000) can be achieved whenever possible by techniques that can help in the movement from different modeling layers to simulation and execution.

Introducing simulation, as opposed to debugging and testing, accompanied by its full power to the UML activity modeling leads to benefits starting with enabling underlying simulators to be employed for studying models during their development life cycle. Such a goal is difficult to achieve without having precise model semantics. The DEVS formalism can be utilized to define these semantics. It is actually a suitable candidate to achieve this objective. Moreover, it defines the notion of time yet more rigorously which can be utilized further, for example for cyber-physical systems and more broadly Internet-of-Things. The defined DEVS specification for the semantics can effectively serve as a basis for the concrete counterparts allowing a wide variety of component-based simulators.

REFERENCES

Eclipse Foundation 2016. “Papyrus Mars release (1.1.3)”. Available at https://eclipse.org/papyrus/.
AUTHOR BIOGRAPHIES

ABDURRAHMAN ALSHAREEF is a Lecturer in the College of Computer and Information Sciences at King Saud University. He is currently pursuing his Ph.D. in Computer Science program in the School of Computing, Informatics and Decision Systems Engineering at Arizona State University (ASU), Tempe, AZ. His research interests lie in the discrete event system models development and simulation, model-driven engineering, software architecture and metamodeling. He can be contacted at alshareef@asu.edu.

HESSAM S. SARJOUGHIAN is an Associate Professor of Computer Science and Computer Engineering in the School of Computing, Informatics, and Decision Systems Engineering (CIDSE) 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 for complexity science, and M&S frameworks/tools. He is the director of the ASU Online Masters of Engineering in Modeling & Simulation program. He can be contacted at hss@asu.edu.