ACTIVITY DIAGRAMS BETWEEN DEVS-BASED MODELING & SIMULATION AND FUML-BASED MODEL EXECUTION

Abdurrahman Alshareef

Information Systems Department
College of Computer and Information Sciences
King Saud University
P.O. Box 2454, Riyadh 11451
Riyadh, Saudi Arabia
ashareef@ksu.edu.sa

Doohwan Kim

RTSync Corp.
6909 W Ray Rd STE 15-107
Chandler, AZ, USA
dhkim@rtsync.com

Chungman Seo

RTSync Corp.
6909 W Ray Rd STE 15-107
Chandler, AZ, USA
cseo@rtsync.com

Bernard P. Zeigler

RTSync Corp.
6909 W Ray Rd STE 15-107
Chandler, AZ, USA
zeigler@rtsync.com

ABSTRACT

Activity Diagrams are prominently used for modeling for both system and software engineering. Their metamodel in the Unified Modeling Language (UML), and the System Modeling Language (SysML), is used to standardize specifying behavioral models in a flow-based manner. However, due to the high-level metamodel, further description is needed to facilitate a precise interpretation of these diagrams in a computational environment. A Discrete Event System Specification (DEVS) was devised to provide an initial simulation of activities and therefore enable more advanced simulations and experiments. Another line of research has been focusing on providing execution for activities according to the Semantics of a Foundational Subset for Executable UML Models. In this work, we examine both approaches highlighting the similarities and differences. We also attempt to examine the capacity of each approach to handle potential complexity that might arise during the model lifespan, especially for modeling System of Systems.

Keywords: Activity Diagrams, DEVS, Model-Based System Engineering, SysML, UML.

1 INTRODUCTION

Activity Diagrams (referred to as ADs or activities) are becoming more central in model-based and model-driven approaches for system and software engineering. Due to their essential and relatively more primitive nodes and actions, it became quite attractive for researchers to develop some precise semantics for them in order to address a major concern, which is the delivery of concrete realizations of the constructed models according to the high-level description and model-driven principles. Due to the encountered interpretation gaps in ADs inherited from the Unified Modeling Language (UML) (OMG 2017), it is necessary to provide

©2020 Society for Modeling & Simulation International (SCS)

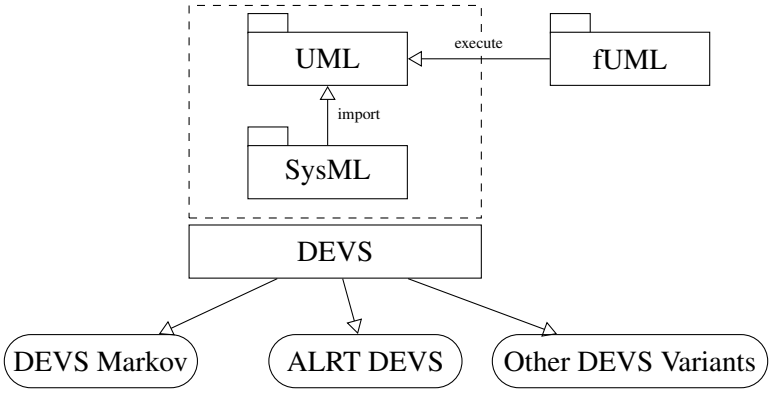


Figure 1: fUML provides an execution model for the UML while DEVS specification underlines SysML and UML while providing the simulation capability and therefore enabling them to undertake different simulations such as DEVS Markov simulation.

additional concrete layers to overcome such an issue. Multiple paths have been taking place. In this paper, we will discuss two of them by which notions of simulation and model execution are the focus. The first one is through Discrete Event System Specification (DEVS)(Zeigler, Muzy, and Kofman 2018), while the other one is through Foundational Subset for Executable UML (fUML) (OMG 2018).

For the first path, the transformation model between the DEVS formalism and the UML has been examined on multiple occasions while sometimes employing Model-Driven Architecture. The System Modeling Language (SysML) (OMG 2019) extends the UML with some system notions to establish a more suited standard toward system engineering tasks. However, it still requires more layers if modelers are interested in execution or simulation for more concrete results. It is challenging to create a transformation between the DEVS formalism, which is mathematical and precise, on the one hand, and the UML and SysML, which is less precise and semi-formal, on the other.

For the second one, the fUML standard has been proposed to primarily serve the UML without direct consideration to its extensions or profiles, including SysML or MARTE (the UML profile for model-driven development of real-time and embedded systems). We mention those two profile examples due to their inadequacy for conducting a simulation as required in a system engineering process. The reason is mainly due to their account of timing for reasons that we will discuss in the related work section. As a result, the fUML model is more aligned toward the UML itself without substantial accounts of the system elements. Such distinctions are crucial from a system standpoint. The resulting execution model in the fUML is more tuned toward a specific programming language with stereotypes and specific action types. For example, the *Read Self* action corresponds to something similar to the keyword *this* in Java programming language.

Our goal in this paper is to examine both approaches and attempt to highlight similarities and differences between a DEVS-based as opposed to an fUML-based approach to propose simulation and execution environments for UML and SysML. Our consideration will be to some extent general with a particular focus on ADs (we will use ADs and activities interchangeably). Figure 1 shows some relationships between all the parts and how to navigate through them in this study. While the focus of both the fUML execution model and the DEVS specification is centralized around the activity node and therefore ADs, they both serve as a basis for UML and SysML as a whole. The role of action is fundamental in dictating the behavioral modeling offered in both languages.

We start with the related work discussion in Section 2 considering both DEVS and fUML approaches for ADs, UML, and SysML. We also give a brief background in Section 3 about ADs and their proposed simulations. Then, we discuss ADs with consideration of some corresponding models to different examples

in different fUML and DEVS environments while demonstrating the modeling of these examples in each approach (Section 4). In Section 5, we highlight our findings about modeling in fUML, DEVS, UML, and SysML with some discussion. We conclude our work in Section 6.

2 RELATED WORK

The endeavor toward simulating SysML and UML at large faces two major obstacles. The first one is the lack of a rigorous timing definition. The second one relates to the issue of semantics in both languages since they generally tend to capture high-level descriptions. Both issues arise only at the simulation and execution stages. They are realized in many research works throughout the past two decades. Störrle and Hausmann (2005) and Damm, Josko, Pnueli, and Votintseva (2005) realize the second, semantics, issue with respect to many modeling constructs, some of which are quite essential such as control nodes. Many other existing works (e.g.,(Partsch et al. 2011)) realize the former, timing, issue while there are fewer attempts to address it using various techniques and approaches. We should note that these issues do not arise in all stages of the design process, especially if the ultimate goal is solely to develop such artifacts and simulation is not intended. However, there is a growing interest in making use of such artifacts in a model-based system engineering (MBSE) methodology supported by fully capable modeling and simulation (M&S) activities referred to as MBSE/M&S in the sequel (Zeigler, Mittal, and Traore 2018).

Since UML and SysML are not formally defined, multiple possible interpretations can be arrived at from a single model. This ambiguity also applies at the other end where multiple and different model abstractions can correspond to a single implementation or some aspects thereof. The existence of many-to-many interpretations has been subject to study by many researchers.

As an example, the use of Petri nets in representing ADs provides a useful insight with respect to the reachability analysis in the context of model checking and verification (Agarwal 2013) after imposing the necessary restrictions on the state space. Although Petri net-like semantics characterize some aspects of ADs (OMG 2017), many expressive features in ADs relative to the Petri net model remain undefined. The Petri net, as a special type of a bipartite graph, is far more restrictive in topology than is an activity. Petri net procedures do not apply to ADs unless a transformation step takes place. Many transformation models have been proposed from ADs to Petri nets (e.g., Störrle and Hausmann (2005), Agarwal (2013)). Although ADs have their own restrictions, they are entirely different from the ones encountered in a Petri net graph.

With respect to simulation, in order to define the relation between the model and the simulator precisely, we must consider the relationship of state and time (Nance 1981). This is generally not achievable solely by specification of fUML since its proposed execution model is timeless or *time agnostic* (OMG 2018). Nevertheless, some works and tools rely on this execution model in order to execute or simulate a UML model (Mohlin 2010, NoMagic 2020, Eclipse Foundation 2019). The fUML execution model accounts for the UML specification. It does not directly target the SysML (OMG 2019), which relatively accounts more for system engineering aspects. The outcome implementations are, therefore, subject to the same rationale underlying the proposal of SysML, which is to serve for system engineering modeling and software-intensive systems therefore accounting for MBSE/M&S as mentioned above. The time base is only defined with a partial order. Such deficiency contributes to the challenging nature of simulating SysML (Nikolaidou et al. 2015). The work of Tatibouët, Cuccuru, Gérard, and Terrier (2014) also highlights this simulation difficulty and ascribes it as generally due to the agnosticism of time in UML, and the lack of precise time definitions in the SysML. Hence, *action*, which is a central element in ADs and the UML at large, is timeless in SysML. Some timing constructs are selectively defined for some modeling constructs, such as time intervals/ticks or the timing associated with *accept event action*. Nevertheless, the lack of an independent time base makes it difficult to simulate the UML or SysML or a diagram thereof as a whole.

Some works consider DEVS as a formalism to provide the capability of simulation for both UML and SysML at large and fewer works consider ADs in particular. Different approaches have taken place for the transformation and integration between the DEVS formalism on the one hand and the UML and SysML on the other. Risco-Martín, De La Cruz, Mittal, and Zeigler (2009) proposed to execute UML models with DEVS via mapping elements of the UML to a counterpart element in DEVS. For example, the proposed metamodel maps the component to the atomic and the coupled model. It also maps the sequence diagram to a coupled model with external and internal events. In other works (Risco-Martın, Mittal, Zeigler, and Jesús 2007, Mooney and Sarjoughian 2009), the proposed metamodel accounts only for a specific diagram, which is statecharts. In (Mooney and Sarjoughian 2009), it proposes statechart notations with consideration to the concept of state in DEVS. ADs as well have been subject to examination concerning its possible relation to DEVS (Özmen and Nutaro 2015, Alshareef, Sarjoughian, and Zarrin 2018). Alshareef and Sarjoughian (2017) propose DEVS as an underlying formalism for activities by DEVS modeling its action, fork, join, decision, merge, flow, and the rest of its elements and use the notion of state as defined in DEVS with inherent timing specification and the underlying time base. In this paper, we focus our comparison for ADs simulation and execution between fUML-based approaches (e.g., Guermazi et al. (2015)) on the one hand and a DEVS-based approach on the other (i.e., Alshareef (2019)).

3 BACKGROUND

The activities metamodel (OMG 2017) primarily consists of nodes and edges with many further specializations, mostly for nodes. An example of object nodes is the parameter which can refer to distribution and defines the arrival, or departure, of inputs, or outputs. Control nodes can be *merge*, *decision*, *join*, *fork*, *initial*, and *final* nodes, each of which describes a different type of control over the flow. The *merge* and *decision* nodes select some incoming or outgoing flows to proceed based on some condition. In SysML (OMG 2019), the likelihood of traversing the outgoing flow of a decision node can be defined with probabilities. The *join* and *fork* nodes synchronize the incoming or outgoing flows waiting for all incoming flows in *join* and resulting in concurrent flows in *fork*. The *initial* and *final* indicates the initiation and termination of an activity. The other important type of node is the action node. It is the super-type of many specialized types of actions in the fUML model.

In the DEVS-based approach, an atomic model specification describes the action as well as the control node. We refer the reader to (Alshareef, Sarjoughian, and Zarrin 2018) for further details about the DEVS formal specification of the action, and four control nodes that are *merge*, *decision*, *join*, and *fork*. The DEVS correspondent to the activity diagram is then the coupled model created based on the structure of the given activity. Stimuli can be received by either a generator component or through an external input coupling resulting in the activation. The Parallel DEVS model (Zeigler, Muzy, and Kofman 2018) is $\langle X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$. X is the set of input events. S is the set of sequential states. Y is the set of output events. δ_{int}, δ_{ext}, and δ_{con} are the internal, external, and confluent transition function, respectively. λ is the output function, and ta is the time advance function. A specification is devised for each type of activity node. Primarily, three atomic models are specified to correspond to *action*, *join* and *fork*, and *merge* and *decision*. These three atomic models mimic the behavior of their corresponding nodes during the simulation according to their ascribing semantics. Other aspects of the models are also captured, such as the flows, I/O parameters, and pins.

4 MODELING SIMPLE INCREMENT EXAMPLE, MULTI-PROCESSING SCHEMES, AND ENGAGEMENT OPERATION

We present different examples and how to use activities to model the examples. The first example in section 4.1 is a simple increment procedure where a number is initialized with some integer value, and then a value of one is added in each iteration. It is a simple procedure in any programming language. We use

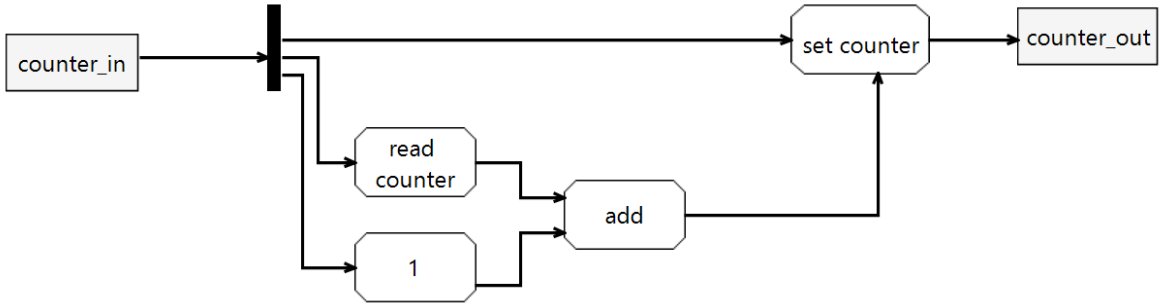


Figure 2: Modeling a simple increment procedure using an activity

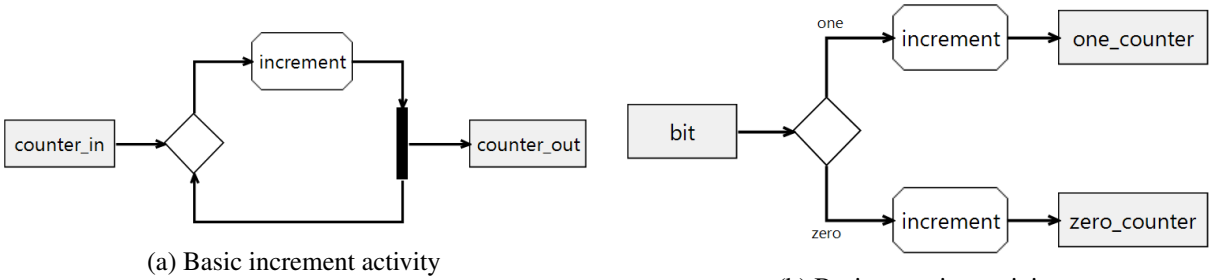
this example, especially to demonstrate the use of the fUML-based approach. We also model it using the DEVS-based approach. The second example in section 4.2 is devised to illustrate the capability of activities to model a dynamic system. This capability has been achieved through having DEVS as an underlying formalism for the activities and UML/SysML. Finally, we demonstrate with a more complex example of an engagement operation of a defense system with a ballistic missile in section 4.3.

4.1 Simple increment example

This example was presented by (Eclipse Foundation 2019) for the fUML-based modeling of activities. The activity consists of an input parameter representing the value being received by the activity. Then, a fork node splits the flow into two outgoing flows. The first flow gets into a value specification action to modify the value by one as to accomplish the increment procedure, while the other flow generates the value one itself, which is to be added to the current value. As soon as the value of one being issued, it gets directed to the value specification action. The addition operation is selected to take place on the received values. The resulting value (counter+1) gets directed then to the output parameter. Figure 2 shows the activity corresponding to this procedure in preparation for it to be executed using MOKA (Eclipse Foundation 2019), which is an execution engine that conforms to the fUML execution model. The same activity can also be simulated in DEVS compliant simulator using the specification presented by Alshareef (2019).

Hence a flow goes from the fork node to *set counter* action. This flow is specified originally to account for the fUML-based execution of this activity since the fUML model categorizes different types of actions. *Set counter* action has the type *Add Structural Feature Value Action*. Therefore, it requires two incoming flows, one of which must carry out the object containing the counter variable, which is the flow coming out from the fork node. The other incoming flow to this action is the one coming from *add* action, which is of type *Call Behavior Action* to call the implementation of the addition procedure.

While the same activity can be modeled and therefore simulated using DEVS specification, it can be made more succinct by focusing only on the modeling aspects of activity and utilizing the simulation protocol as a means for the operational semantics. We devise an activity for the increment procedure, and we also devise another activity for counting the incoming ones and zeros as for a counter model (Figure 3). The reason for devising those two activities is to demonstrate some level of succinctness that can result from having the DEVS underlying specification and focusing on the modeling aspects while leaving as much of the platform-related specifics. The codes corresponding to these activities are generated and therefore simulated in DEVS-Suite (ACIMS 2019) and MS4 Me environments (MS4 Systems 2018).



(a) Basic increment activity

(b) Basic counting activity

Figure 3: Demonstrating two different activities and structures for two different models

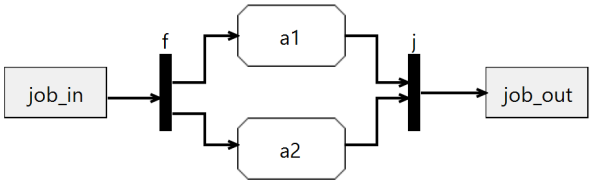
4.2 Modeling multi-processing schemes

Alshareef and Sarjoughian (2018) presented the use of activities as an abstraction of different processing archetype architectures. They devise different activities to model different processing regimes such as pipeline and divide and conquer. A variety of simulations are obtained and carried out accordingly with some observations regarding the parallelism. In addition to the previously discussed simulations, we will discuss the simulation of the architectures with DEVS Markov simulation in this paper. Having the underlying DEVS specification enables further examination of the parallelism semantics akin to the activity modeling approach. Fork and Join nodes are central in ADs. Thus, an account of the resulting parallel flows is a reasonable expectation from any environment by which simulation of ADs is carried out. Also, an account of the fork-join scheme is necessary to facilitate the progression of the flow in a parallel manner. We refer the reader to the previous paper (Alshareef and Sarjoughian 2018) for more details about the specification and simulation of the architectures mentioned above. In this paper, we focus on the DEVS Markov simulation of activities that are being devised as an abstraction for different multi-processing schemes.

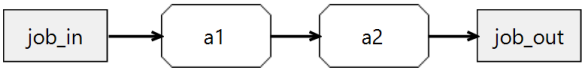
4.2.1 DEVS Markov simulation of activities

DEVS Markov models (Seo, Zeigler, and Kim 2018) are DEVS models with transition probabilities and time advances based on Markov property. The actions of activities are equipped with state; therefore, the dynamic changes that they cause are precisely ordered on the time base. The actions in the multi-processing architectures are responsible for carrying out the task of processing a job or some part thereof. State transitions from a passive state to an active state and vice versa describe the action engagement with the job. In an earlier demonstration, fixed positive timings are assumed in the initially generated code for the simulation experiment. However, this timing can be changed by the modeler in the DEVS conforming environment, including the use of probabilistic modeling and distributions. Here, we generate models for simulation in MS4 Me environment, as shown in Figure 4, where the exponential distribution and its mean are specified. However, these selections can be changed by the modeler in MS4 Me environment. A Continuous-Time Markov (CTM) model determines the next transition upon the arrival of some external event. As mentioned earlier, an action can be in a passive or active state (note: the state set need not be restricted to this set). Moreover, the time advance function spans over the continuous-time base indicating the beginning of processing, as well as its termination, can take place at any time instant.

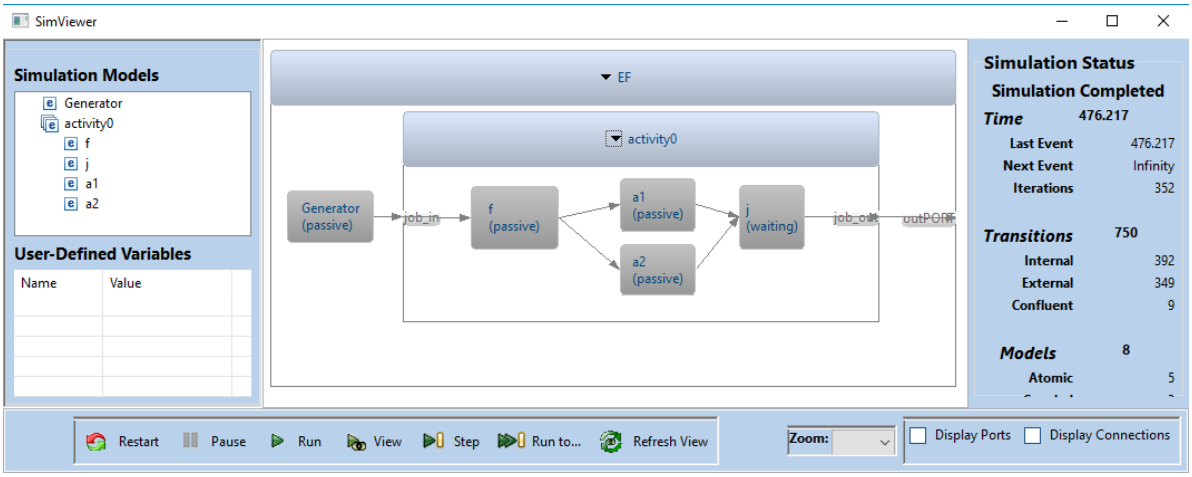
Also, multiple actions can be in an active state simultaneously as the fork-join scheme presented in Figure 4c, which aligns with the ADs semantics as specified in UML as well as SysML. Figure 4c and 4d shows the view after the simulation completion with a hundred jobs generated uniformly in each. Existing tools implicitly impose an order on parallel actions (Mohlin 2010) or requires a multi-threading platform to do so (Eclipse Foundation 2019).



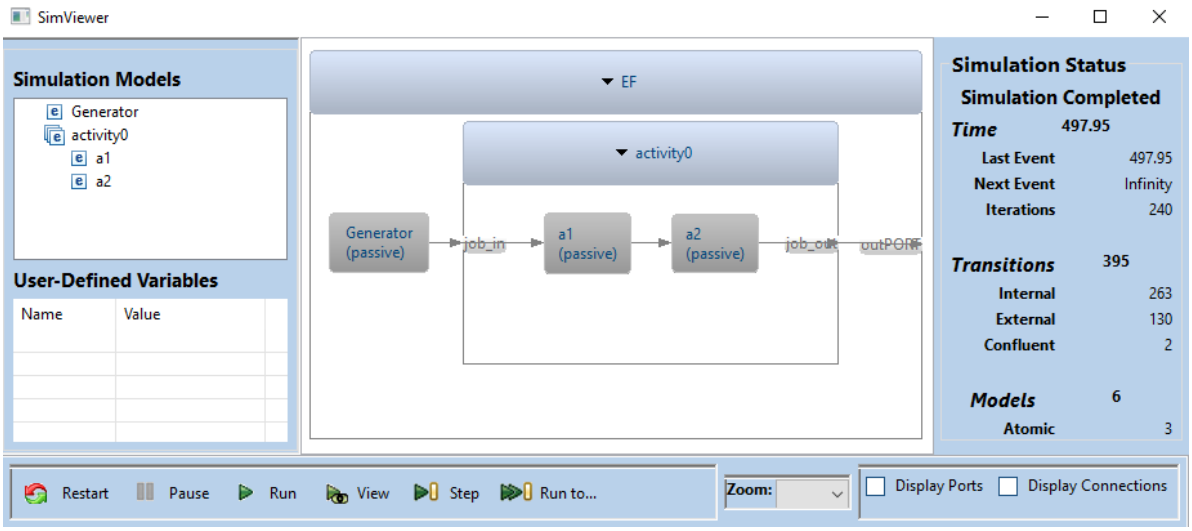
(a) An activity for modeling fork-join multi-processing scheme



(b) An activity for modeling a simple pipeline



(c) Simulation view after completion for generated DEVS Markov models corresponding to the activity in Figure 4a



(d) Simulation view after completion for generated DEVS Markov models corresponding to the activity in Figure 4b

Figure 4: Demonstrating two different activities and their structures for two different models

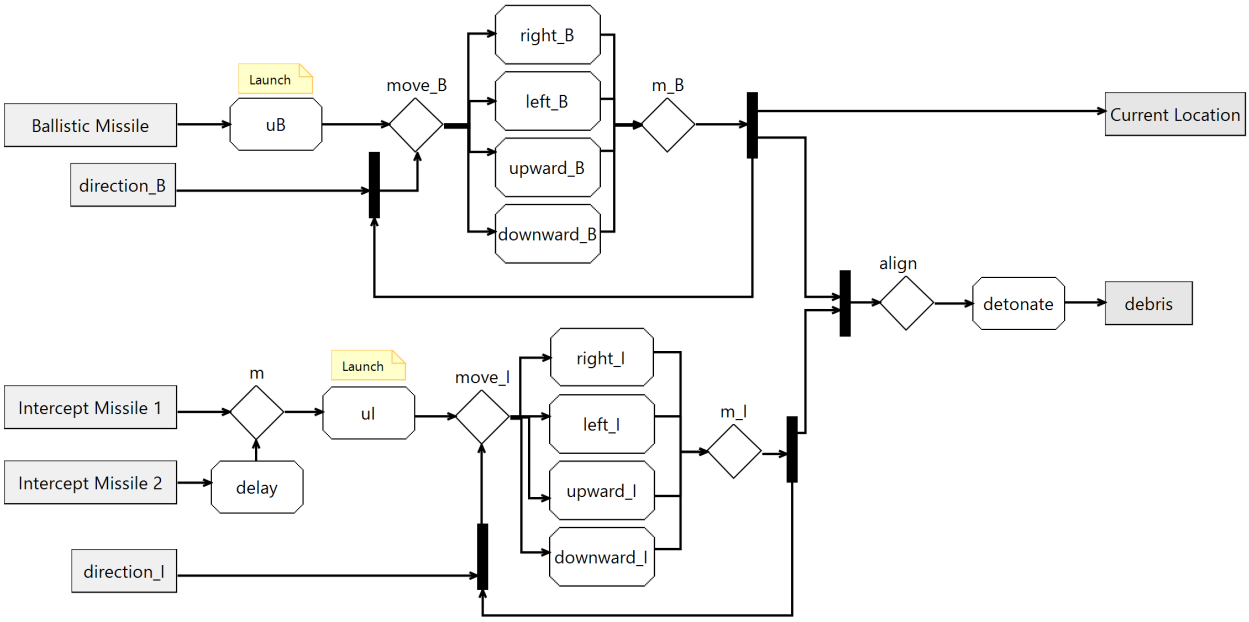


Figure 5: Modeling the engagement activity of the MIM-104 Patriot System with a tactical ballistic missile

4.3 Modeling engagement operation with ballistic missile

Now, we discuss a relatively more complex example. Figure 5 demonstrates the activity modeling using a typical engagement scenario in the MIM-104 Patriot System with a ballistic missile. The activity has five input parameters. Three of them represent the missiles. The first input parameter is the ballistic missile. The other two missiles correspond to the two interceptor missiles that are going to be both used to engage with a single ballistic missile. The first one detonates the ballistic missile while the second one locates debris and attack in a similar manner. The first action for each missile is to launch the missile upward. The second interceptor missile encounters some delay before the launch. Each missile encounters the four actions to represent movements while possibly cycling through the direction adjustments constantly modeled using the join and fork as control nodes for that purpose. The decision node *align* represents the alignment precision of the missile on the interception trajectory before the detonation and destruction of the missile.

The activity diagram in Figure 5 expresses some aspects of the engagement operation using the parameters, actions, selection, and synchronizing schemes in addition to the different flows connecting them. When the DEVS formalism underlies the flow of the various activity nodes and edges, the simulator can provide the expected behavior for the corresponding DEVS model accurately. Therefore, we can use that as a framework to examine different tactics with large different assignments that can be formulated in various ways using the set of DEVS specifications developed to correspond to the activity elements. The nodes ordering and linkages can be rearranged in various ways, whether for producing similar behaviors or entirely different ones. For example, the activity in Figure 4b can be rearranged using further control constructs to represent the facilitation of the architecture with different dynamics that may exhibit with relatively more complex pipeline architectures (Alshareef and Sarjoughian 2018).

Moreover, the specified correspondence enables various timing and parallel examinations in a rigorous setting without imposing restrictions on the modeling process, as opposed to model checking, for example. The simultaneous arrival of input events, state transitions, input parameters, and the likes pose enormous challenges for the underlying simulation environments, among which the computational complexity, one of the motivations for using DEVS Markov. However, the amalgamation of concepts across DEVS and activity

diagrams is promising due to their accounts for handling parallelism and concurrency. An assimilation of the domain artifacts would have the potential to open up a wide range of possibilities for modeling defense strategies in dealing with new threats. Currently, the code generation facility generates the code of DEVS models as well as for DEVS Markov models. The state transition selection is probabilistic in the latter for the action of the activity. The outgoing flows of the decision node in the activity can also encounter a probabilistic selection from which one outgoing flow should proceed.

The generated code is organized into three packages. The first package is for the simulation in DEVS-Suite with a fixed time assignment for the next state transition. The second one is also generated in a similar manner but for the simulation in MS4 Me environment. While the third package is generated for the DEVS Markov simulation in MS4 Me environment as well, where the probabilistic state transition is set for the state transition in the action of the activity and also for the outgoing flows from the decision node. After the code generation, the initial simulation is ready to be conducted. Changes, improvements, and observations can take place within the simulation environment such as manipulation of the random variables and probability distribution as well as observing the simulation through the tool's interface, the transducer, or the generated time series. Further refinements can also take place on the model to the extent of the hardware-level design with more complex representation such as in (Zeigler et al. 2020).

4.4 Model simulation versus execution

The terms simulation and execution invoke different understandings in the context of model-based design and modeling and simulation. Some authors use the term model execution or execution semantics (Tatibouët et al. 2014) and therefore attempt to formalize it for some UML profiles. Nikolaidou et al. (2015) use executable simulation models to refer to codes corresponding to the SysML model. In our view, UML and SysML as modeling languages are used in conjunction with the DEVS modeling formalism in order to enrich the modeling process with notations, principles, and simulation engines to concretely translate the developed models into some form of analyzable and observable outcomes, results, and data.

5 BETWEEN FUML AND DEVS FOR UML AND SYSML

The fUML-based execution of the activity in Figure 2 is described now. Upon the receipt of the token via the input parameter $counter_{in}$, the outgoing flow from this parameter to the fork node is enabled. The token transfers to the fork node, which in turn enables all the three outgoing flows from it. However, the action *set counter* cannot be enabled because it expects another incoming flow from another source. Therefore, only action *read counter* and action *1* are enabled. They can execute in any order but not in parallel due to the limitation mentioned above in the fUML execution model. After executing both, the action *add* is enabled which then enables the action *set counter*. Such execution can be described as a discrete time model in the following 7-tuple where each tuple corresponds to a single activity node $\langle counter_{in}, fork, setcounter, readcounter, one, add, counter_{out} \rangle$ with values zero or one. The value one indicates the existence of a token in the corresponding element. The following 7-tuple corresponds to the initial state of the execution $\langle 1,0,0,0,0,0,0 \rangle$ where none of the nodes has a token upon the arrival of the input except the input parameter node. The existing implementations of the fUML model prevent having multiple tokens at different nodes simultaneously and, therefore, prevent concurrent flows. The state $\langle 0,0,1,1,0,0,0 \rangle$ complies with the semantics of the activities; however, it is not reachable in the existing executions posing a restriction on the discrete time model. Moreover, an fUML model may not necessarily have any time progression constructs such as *accept event action*, which leads to the well-known Zeno phenomenon causing infinite state transitions in a finite time.

Concurrent flows are central to the semantics of activities (ADs). The DEVS-based simulation accounts for such semantics by removing the restriction on the state space and allowing parallel actions/nodes to proceed in various orders conforming to the semantics. Some restrictions on the state space can be kept in place for specific benefits, such as making verification attainable. However, imposing them in the earlier stages can render some limitations on the resulting models. While the state space in DEVS is unconstrained, Gholami and Sarjoughian (2017) and Yacoub et al. (2017) described ways in which such restrictions can take place on the state space allowing for model checking and verification after that and resulting in combined and more integrative approaches of DEVS and model checking. Conversely, a variety of commonly used simulations can be enabled, such as Markov DEVS and Action Level Real-Time (ALRT) DEVS (Sarjoughian and Gholami 2015) both of which could have the activity (ADs) as an entry point to the modeling process and a significant link to the model-based design and languages. We summarize the distinctions between the fUML-based and DEVS-based approach activity in Table 1.

Table 1: The execution and simulation of activities are achieved through fUML or DEVS with the following major distinctions between the two approaches.

	fUML-based execution	DEVS modeling and simulation
Formalism	Discrete Time model with step-wise execution	Discrete Event
Time	Implicit isomorphic to integer time base in single thread	Real time base
Progression	Undefined	Legitimacy property
I/O	I/O parameters and pins	I/O ports
Modularity	Calling behavior actions	Modular/Closure Under Coupling
State space	Finite Deterministic	Deterministic/Stochastic
Operational semantics	Execution model	Simulation protocol
Weaknesses	Readability and scalability	Unconstrained state space
Suitable for	Software development and interfaces	Simulation modeling/System of Systems
Parallelism	Sequential	Parallel simulation

The System Entity Structure (SES) (Zeigler, Muzy, and Kofman 2018) can be employed to organize the model classes. We look into the created models to simulate ADs from the SES viewpoint in order to characterize them in terms of a time base, state set, external event set, and transition function mapping. The underlying formalism, time, progression, I/O, and state space, are observed in Table 1 for the resulting models from both approaches, that is, the acquired models from the fUML-based and DEVS-based approach. Other classes and sub-classes can be formed by choice of elements. Some choices can be left unspecified.

6 CONCLUSION

We have examined the fUML and DEVS approaches to facilitating a more precise interpretation of UML activity diagrams in a computational environment. We then highlighted the similarities and differences in terms of their expressive potential and computational properties. We also examined the capability of each approach to handle potential complexity that might arise during the model lifespan, especially for modeling System of Systems. In the context of MBSE/M&S we showed that mapping ADs to DEVS models, especially to DEVS Markov models, opens the door to a full range of modeling and simulation capabilities for capturing the dynamics of complex systems as well as performing the necessary simulations on various computational platforms from workstations to cloud-based environments.

REFERENCES

ACIMS 2019. "DEVS-Suite Simulator version 5.0.0". Available at http://ms4systems.com/pages/ms4me.php (Accessed April 1, 2020).

Agarwal, B. 2013. "Transformation of UML activity diagrams into Petri nets for verification purposes". *International Journal Of Engineering And Computer Science* vol. 2 (3), pp. 798–805.

Alshareef, A. 2019. "Activity Specification for Time-based Discrete Event Simulation Models". *Ph.D. Dissertation, Arizona State University*.

Alshareef, A., and H. S. Sarjoughian. 2017. "DEVS specification for modeling and simulation of the UML activities". In *Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering*.

Alshareef, A., and H. S. Sarjoughian. 2018. "Parallelism semantics in modeling activities". In *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences*.

Alshareef, A., H. S. Sarjoughian, and B. Zarrin. 2018. "Activity-based DEVS modeling". *Simulation Modelling Practice and Theory* vol. 82, pp. 116–131.

Damm, W., B. Josko, A. Pnueli, and A. Votintseva. 2005. "A discrete-time UML semantics for concurrency and communication in safety-critical applications". *Science of Computer Programming* vol. 55 (1-3).

Eclipse Foundation 2019. "Moka release (4.0.0) for Eclipse Papyrus release (4.7.0)". Available at https://eclipse.org/papyrus/ (Accessed April 1, 2020).

Gholami, S., and H. S. Sarjoughian. 2017. "Modeling and verification of network-on-chip using constrained-DEVS". In *Proceedings of the Symposium on Theory of Modeling & Simulation*, pp. 1–12.

Guermazi, S., J. Tatibouet, A. Cuccuru, S. Dhouib, S. Gérard, and E. Seidewitz. 2015. "Executable modeling with fUML and ALF in papyrus: Tooling and experiments". *strategies* vol. 11, pp. 12.

Mohlin, M. 2010. "Model simulation in rational software architect: simulating UML models". *Cupertino, CA: IBM*.

Mooney, J., and H. S. Sarjoughian. 2009. "A framework for executable UML models.". In *SpringSim*.

MS4 Systems 2018. "MS4 Me Simulator version 3.0". Available at http://ms4systems.com/pages/ms4me.php (Accessed April 1, 2020).

Nance, R. E. 1981. "The time and state relationships in simulation modeling". *Communications of the ACM* vol. 24 (4), pp. 173–179.

Nikolaidou, M., G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos. 2015. "Simulating SysML models: Overview and challenges". In *2015 10th System of Systems Engineering Conference (SoSE)*.

NoMagic 2020. "Cameo Systems Modeler". Available at https://www.nomagic.com/products/cameo-systems-modeler (Accessed April 1, 2020).

OMG 2017. "Unified Modeling Language version 2.5.1".

OMG 2018. "Semantics of a Foundational Subset for Executable UML Models (fUML) version 1.4".

OMG 2019. "System Modeling Language version 1.6".

Özmen, Ö., and J. Nutaro. 2015. "Activity diagrams for DEVS models: a case study modeling health care behavior (WIP)". In *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pp. 129–134.

Partsch, H., M. Dausend, D. Gessenharter, J. Kohlmeyer, and A. Raschke. 2011. "From Formal Semantics to Executable Models: A Pragmatic Approach to Model-Driven Development.". *Int. J. Software and Informatics* vol. 5 (1-2), pp. 291–312.

Risco-Martín, J. L., J. M. De La Cruz, S. Mittal, and B. P. Zeigler. 2009. "eUDEVS: Executable UML with DEVS theory of modeling and simulation". *Simulation* vol. 85 (11-12), pp. 750–777.

Risco-Martın, J. L., S. Mittal, B. P. Zeigler, and M. Jesús. 2007. "From UML state charts to DEVS state machines using XML". In *workshop on multi-paradigm modeling: concepts and tools, Nashville, TN*.

Sarjoughian, H. S., and S. Gholami. 2015. "Action-level real-time DEVS modeling and simulation". *Simulation* vol. 91 (10), pp. 869–887.

Seo, C., B. P. Zeigler, and D. Kim. 2018. "DEVS markov modeling and simulation: formal definition and implementation". In *Proceedings of the 4th ACM Inter. Conference of for Engineering and Sciences*.

Störrle, H., and J. H. Hausmann. 2005. "Towards a formal semantics of UML 2.0 activities". *Software Engineering 2005*.

Tatibouët, J., A. Cuccuru, S. Gérard, and F. Terrier. 2014. "Formalizing execution semantics of UML profiles with fUML models". In *International Conference on Model Driven Engineering Languages and Systems*, pp. 133–148. Springer.

Yacoub, A., M. E. A. Hamri, C. Frydman, C. Seo, and B. P. Zeigler. 2017. "DEv-PROMELA: an extension of PROMELA for the modelling, simulation and verification of discrete-event systems". *International Journal of Simulation and Process Modelling* vol. 12 (3-4), pp. 313–327.

Zeigler, B. P., N. Keller, D. Kim, C. Anderson, and J. Ceney. 2020. "Supporting the Reuse of Algorithmic Simulation Models". In *20-MDA-10481*.

Zeigler, B. P., S. Mittal, and M. K. Traore. 2018. "MBSE with/out Simulation: State of the Art and Way Forward". *Systems* vol. 6 (4), pp. 40.

Zeigler, B. P., A. Muzy, and E. Kofman. 2018. *Theory of modeling and simulation: discrete event and iterative system computational foundations*. Academic press.

AUTHOR BIOGRAPHIES

ABDURRAHMAN ALSHAREEF is an Assistant Professor in the Information Systems Department at the College of Computer and Information Sciences, King Saud University, and a member of Arizona Center for Integrative Modeling & Simulation (ACIMS). He holds a PhD in Computer Science from Arizona State University. His research interests lie in modeling and simulation, metamodeling, and behavioral modeling. His email address is ashareef@ksu.edu.sa.

DOOHWAN KIM is the founder and president of RTSync Corp., which specializes in Predictive Analytics and Model-Based System Engineering based on DEVS M&S technology. Dr. Kim has been involved in the design, development, and delivery of the advanced M&S solutions for highly complex real world information science and engineering problems. He received his Ph.D. degree from the Department of Electrical and Computer Engineering of the University of Arizona in 1996. His email address is dhkim@rtsync.com.

CHUNGMAN SEO is a senior research engineer at MS4Systems Company and a member of Arizona Center for Integrative Modeling & Simulation (ACIMS). He received his Ph.D. in Electrical and Computer Engineering from The University of Arizona in 2009. His research includes MS4 Me Product, DEVS based testing environment with DEVS inverse model, Model-Based System Engineering with DEVS , and DEVS interoperability. His email address is cseo@rtsync.com.

BERNARD ZEIGLER is a Professor Emeritus from the University of Arizona and Chief Scientist at RTSync Corp.. He is a Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). He is recognized by Simulation Archive as a Computer Simulation Pioneer. His biography appears in Wikipedia. His email address is zeigler@rtsync.com.