International Journal of Modeling, Simulation,
and Scientific Computing
(2022) 2230001 (16 pages)

DOI: 10.1142/S1793962322300011

DEVS and MBSE: A review

Bernard P. Zeigler

Department of Electrical & Computer Engineering
College of Engineering, The University of Arizona
Tucson, AZ 85721, USA

RTSync Corp., Chandler, AZ 85226, USA
zeigler@rtsync.com

Received 18 July 2021
Accepted 19 September 2021
Published

We review Discrete-Event system Specification (DEVS) in the context of Model-based Systems Engineering (MBSE) and discuss an application of DEVS methodology to MBSE. We outline support for an envisioned MBSE development cycle of DEVS top-to-bottom MBSE capability and offer an example of mapping UML activity diagrams into executable activity-based DEVS models. We close with conclusions and future research directions.

Keywords: discrete-event system specification; DEVS; model-based systems engineering; MBSE; UML activity diagrams; homomorphisms; system design; activity-based models.

1. Introduction

Model-based Systems Engineering (MBSE) refers to the trend to use models systematically throughout the design process. MBSE has been struggling to find a way to connect the blueprint models that describe the initial architecture with ways to check and evaluate these high-level plans. Simulation has become the preferred means to support this goal. The Discrete-Event system Specification (DEVS) is a model-based way to perform simulation and a natural enabler of simulation-based MBSE. In this paper, we first review DEVS in the context of MBSE, discussing the DEVS Formalism, associated modeling and simulation (M&S) environments, the hierarchy of system specifications, the DEVS Simulation Protocol, and a timeline history of some key DEVS Developments. As an illustration of application of DEVS methodology to MBSE, we discuss the homomorphic implementation of DEVS-like systems including the hierarchy of system specification morphisms and an application to the design of simulation systems for "as-is" software. The main contribution is discussion of DEVS-based capabilities and tools for MBSE. We outline support for an envisioned MBSE development cycle with DEVS top-to-bottom

MBSE capability. As an example, we discuss mapping UML activity diagrams into executable activity-based DEVS models and further elaborated into a family of simulatable architectural variants. We close with conclusions and future research directions.

2. Review of DEVS in the MBSE Context

2.1. *DEVS formalism*

The DEVS Formalism was developed on the firm mathematical systems theory foundation of Wymore[1] and others and constitutes a well-founded computational basis for systems theory-based modeling and simulation needed by today's digital engineering. The mathematical theory helps ensure reliable system implementation and accurate simulation time management. The DEVS hierarchical construction methodology supports the complex system development needed for correct cyber-physical system M&S and downstream implementation. A DEVS model formalization specifies a model's inputs, states, and outputs in a manner similar to a finite state automaton. However, a key difference is that the formal structure includes a time-advance function which allows it to represent discrete-event systems as well as simulated continuous components. Due to its universality of realization,[2] any real-world system, or System of Systems (SoS), can be modeled in DEVS and simulated in a DEVS-compliant computational platform.

We briefly state and stress some key aspects of DEVS. The DEVS Formalism formalizes what a model is, what it must contain, and what it does not contain (for instance, experimentation and simulation control parameters are not contained in the model). Moreover, DEVS is universal and unique for discrete-event system models. Any system that accepts events as inputs over time and generates events as outputs over time is equivalent to a DEVS model. With DEVS, a model of a large system can be decomposed into smaller component models with couplings between them. The DEVS Formalism defines two kinds of models: (i) atomic models that represent the basic models providing specifications for the dynamics of the system components; and (ii) coupled models that describe how to couple several component models (which can be atomic or coupled models) together to form a new model. This hierarchical construction stems from the proof that a coupled model behaves like an atomic model due to DEVS Formalism's closure under coupling which is strongly linked to systems being well defined.[2]

An atomic DEVS model can be considered as an automaton with a set of states and transition functions allowing the state to change when an event occurs or due to the passage of time. When no events occur, the state of the atomic model is updated by the internal transition function upon expiration of its lifetime. When an external event occurs, the atomic model intercepts it and makes a change in state by applying its external transition function. The lifetime of a state is determined by a time-advance function. Each state change can produce output messages via

the output function. Examples of such DEVS models, including atomic and coupled models, can be found in Ref. 2.

An abstract simulator algorithm is associated with the DEVS Formalism in order to execute the structure of a model to generate its behavior. This technology-free abstract simulator standard enables DEVS models to be simulated on multiple different execution platforms, including those on desktops (for development) and those on high-performance platforms (such as Clusters or High-Performance Computers).

The DEVS Formalism includes continuous, parallel, coupled, networked, and Markov (stochastic) modeling. The Iterative System Specification and its relation to the DEVS Formalism enable defining various continuous, discrete, and hybrid models for which simulations are possible. The concept of system coupling has been proven within DEVS to allow for coupling of both discrete and continuous systems through closure under coupling. The Iterative Specification of DEVS also allows for use of morphisms to maintain consistency and control deviation throughout the specification of the various system models.[2]

2.2. *M&S environment background*

There are many commercial M&S software environments that cover a large variety of application domains. However, general-purpose environments are relatively rare. Here, we review MS4 Me[3,4] which is an environment to design general systems as well as SoS based on systems theory.[5,6] MS4 Me modeling software uses the DEVS modeling formalism and *System Entity Structure* (*SES*)[7] ontology to support model composability. MS4 Me supports the collaboration of domain experts and modelers in both top–down and bottom–up system constructions. The state diagram designer supports graphical specification of atomic models and is automatically (and reversibly) converted to constrained natural language text. This text is translated into a Java atomic model class and compiled to execute in the MS4 Me execution environment based on the DEVS abstract simulator. An atomic DEVS model can be constructed within a constrained natural language using constructs such as time advance, input/output ports, state transitions, internal transitions, external transitions, and output specification. However, a model expressed with limited natural language semantics cannot specify a model's detailed behavior. To overcome this problem, the DEVS natural language file introduces tag blocks which enclose actual (Java) computer code to be inserted in specific locations within the Java class file, e.g., within the characteristic functions of the DEVS Java model. Concepts such as these tag blocks facilitate the inclusion of procedural information into atomic model declarative specifications.

2.3. *Hierarchy of system specifications*

Table 1 identifies four basic levels of system specification forming a Systems Specification Hierarchy and informally describes the incremental knowledge of system

structure gained at each level. The fourth column shows how the levels are applied to *DEVS-like systems*, a class of systems we will be referring to later.

As in Table 2, orthogonal to the level at which a system is specified is the subclass of systems in which the model resides where the most common subclasses are spanned by the modeling formalisms shown in the table. Note that a model may be designated to lie at a certain level of system specification based on whether it is presented in a variety of ways such as data at the I/O Behavior level, behavior generation instructions in the form of a state diagram, or in the form of interacting components, etc. The system specification formalism of models represents models expressed in types of simulation languages as instances of the basic types of system specification: DESS, DTSS, and DEVS. The table shows that such system specifications can occur at any of the levels of specification. In particular, three types in common use (Discrete-Event Simulation, Agent-based Modeling, and Continuous-time Modeling) are included in the basic DESS, DTSS, and DEVS system specifications of TMS (following, e.g., Ref. 8). Moreover, the wider capability of the Iterative System Specification formalism shows how the class of hybrid simulation models is covered.[2]

2.4. *DEVS simulation protocol*

The DEVS Simulation Protocol is a general distributed simulation protocol that prescribes specific mechanisms for:

- declaring the participants in the simulation (component models = federates);
- declaring how federates exchange data;
- executing an iterative cycle that controls how time advances (time management);
- determines when federates exchange messages (data exchange management);
- determines when federates do internal state updating (state update management).

The protocol guarantees correct simulation in the sense that if the federates are DEVS models then the federation is also a well-defined DEVS coupled model. Distinct from the High-Level Architecture (an IEEE standard for distributed simulation), the DEVS Protocol prescribes specific time, data exchange, and state update management processes. These benefits derive from the fact that based on DEVS only a single set of services is needed for all models obviating the need to align divergent services allowed by the looser HLA standard. Moreover, these benefits do not imply undue performance degradation. The Parallel DEVS Simulation Protocol provides close to the best possible performance except possibly where activity is very low or coupling among components is very small.[9] There are numerous implementations of DEVS simulators (see the list by Wainer,[10] Franceschini *et al.*,[11] and Van Tendeloo and Vangheluwe[12]). In particular, ADEVS[13,14] is distinguished by its support for both discrete-event and continuous dynamic systems, both of which are simulated within the DEVS framework. This gives it the capability to simulate

Table 1. Informal description of the levels of system specification, e.g., DEVS-like systems.

Level	Specification name	What we know at this level	DEVS-like systems
0	I/O Frame	How to stimulate the system with inputs; what variables to measure; and how to observe them over a time base?	The input set, X, and output set, Y, are the names of events that occur at discrete instants of time. DEVS(Z) refers to the set of all discrete-event segments with values in the event Z.
1	I/O Relation	Time-indexed data collected from a source system; consists of input/output pairs.	Pairs of input and output time segments from the cross-product of DEVS(X) and DEVS(Y).
3	I/O System	How states are affected by inputs; given a state and an input what is the state after the input stimulus is over; and what output event is generated by a state?	System that when given an input segment in DEVS(X) and an initial state generates an output segment in DEVS(Y) (more discussion in text).
6	Coupled System (Network of Systems)	Components and how they are coupled together. The components can be specified at lower levels or can even be structure systems themselves — leading to hierarchical structure.	Model specified as a composition of DEVS-like components in the same manner that coupled models of DEVS components are constructed.

Table 2. System specification levels and types of framework for the simulation model.

System Specification/Level of Specification	Differential Equation System Specification (DESS)	Discrete Time System Specification (DTSS)	Discrete Event System Specification (DEVS)
Observation Frame	✓	✓	✓
I/O Behaviour	✓	✓	✓
I/O Function	✓	✓	✓
State Transition	✓	✓	✓
Coupled Component	✓	✓	✓

hybrid systems involving the interaction of subsystems characterized by discrete-event dynamics (e.g., communication networks and command-and-control systems) and continuous, physical dynamics (e.g., the trajectories of ballistic missiles and their interceptors).

2.5. *Timeline history of some key DEVS developments*

To summarize, DEVS can be considered as a universal computational formalism for systems.[15] Some of the milestones in its thread of development are listed in Table 3.

Table 3. A historical retrospective on developments including refinements, elaborations, and extensions of the DEVS Formalism with associated references.

Classic DEVS[16]	A formalism for modeling and analysis of discrete-event systems can be seen as an extension of the Moore machine that associates a lifespan with each state and provides a hierarchical concept with an operation, called coupling, based on Wymore's[1] systems theory
Parallel DEVS[17]	Revises the classic DEVS Formalism to distinguish between transition collisions and ordinary external events in the external transition function of DEVS models, extends the modeling capability of the collisions
Hierarchical, Modular DEVS[18]	Implemented DEVS in the object-oriented programming (OOP) and modular programming paradigms[10,12]
System Entity Structure[19]	A structural knowledge representation scheme that contains knowledge of decomposition, taxonomy, and coupling of a system
Dynamic Structure DEVS[20]	Enables representing systems that are able to undergo structural change
DEV & DESS (Discrete-Event and Differential Equation System Specification)[21]	A formalism for combined discrete–continuous modeling which based on system theoretical combines the three system specification formalisms — Differential Equation, Discrete Time, and Discrete Event System Specification formalisms
Quantized State Systems[22]	Dynamical systems are continuous-time systems where the variable trajectories are piecewise constant and can be exactly represented and simulated by DEVS

Table 3. (*Continued*)

GDEVS (Generalized DEVS)[23]	Organizes trajectories through piecewise polynomial segments utilizing arbitrary polynomial functions to achieve higher accuracies in modeling continuous processes as discrete-event abstractions
Modelica & DEVS[24]	Transforms Modelica continuous models into DEVS, thus supporting models with state and time events that comprise differential-algebraic systems with high index
Finite Deterministic DEVS[25]	A powerful subclass of DEVS developed to teach the basics of DEVS that has become the basis implementations of symbolic and graphical platforms for full-capability DEVS
Iterative System Specification[2]	General Mathematical System formalism enables defining various continuous, discrete, and hybrid models extending Wymore's systems theory.[26] Enables DEVS to represent general systems with controlled accuracy

3. Homomorphic Implementation of DEVS-Like Systems

Homomorphisms play a major role in Wymore's systems engineering theory,[26] but so far have not enjoyed corresponding application within the MBSE community. Furthermore, the origin of DEVS as a way to specify a subclass of Wymore's systems definition suggests that employing the homomorphism tools developed in the DEVS context should be applicable to Wymore's systems design theory and MBSE.[26]

Here, we illustrate how DEVS supports using homomorphisms to link system specifications at the various design levels of system specification reviewed in Table 1. We consider design of a simulation environment for SoS models in which one of the components is a software system that is executed "as is" rather than represented as a DEVS component model as are the others. This is interesting because the software component can be tested within the simulated environment established by the others without the development costs incurred in creating a discrete-event model, as well as the risk involved in deriving an inaccurate abstraction. The concept we discuss is to implement a simulation platform to generate the behavior of such model compositions using the DEVS Distributed Simulation Protocol while paying special attention to the "as-is" component. While it is not a conventional discrete-event model, it must still interact with others as if it were DEVS-compliant. In particular, the progress in time of the "as-is" component depends on the execution of its host computer and not on the time scale of the overall simulation. An important instance of such an arrangement occurs when the host is a virtual computer platform and may execute the software at a rate depending on its thread scheduling algorithms. The problem to be illustrated is to establish a homomorphism that provides conditions under which the composite model can be made to be correctly executed on the simulation platform.

3.1. *Hierarchy of system specification morphisms*

To illustrate how homomorphisms apply to the design problem just stated, we first review the hierarchy of system morphisms. The essence of modeling lies in establishing relations between pairs of system descriptions. The Systems Specification Hierarchy of Table 1 is a useful starting point for defining and organizing such model-to-model relationships. The general concepts of homomorphism and isomorphism relate system models at the same level of specification. Corresponding to each of the various levels at which a system may be known, described, or specified, is a relation appropriate to a pair of systems specified at that level. We call such a relation a preservation relation or system morphism because it establishes a correspondence between a pair of systems whereby the features of one system are preserved in the other. Morphisms appropriate to each level of system specification are defined such that higher-level morphisms imply lower-level morphisms. This means that a morphism which preserves the structural features of one system in another system at one level, also preserves its features at all lower levels. Readers can refer to Ref. 2 for a detailed exposition.

3.2. *Application to the design of simulation systems for "as-is" software*

Having reviewed the concept of morphism, we return to the application to the simulation of "as-is" software. As the first step, we model the nonconforming component as a DEVS-like system as mentioned in Table 1. DEVS-like systems are systems whose input and output interfaces are event-like as illustrated in Fig. 1, the systems respond to input segments that are time-indexed discrete events and likewise produce output time segments of the same form. The mathematical representation of DEVS-like systems in Chap. 18 of Ref. 2 allows us to infer that they can respond to external events by immediately changing state and subsequently tracing an input-free state trajectory until a next event occurs. The next event can be another external event occurring later or the generation of an output event. This representation allows us to translate the defining elements of any DEVS-like system into the basic elements of a DEVS.

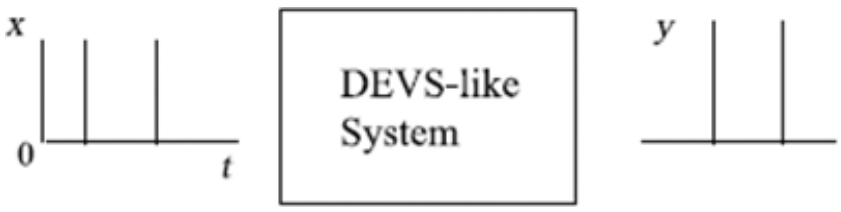

Fig. 1. Pictorial representation of a DEVS-like system with input events x, output events y, occurring on a continuous-time base.

Thus, by assuming the "as-is" model is a DEVS-like system we can represent it as a DEVS model using the translation from its structural elements to the requisite DEVS elements. We can then specialize a known type of homomorphism for a pair of DEVS models to derive conditions that assure that the "as-is" model can be faithfully represented by the virtualized execution. Application of this homomorphism to the DEVS Simulation Protocol that governs the desired simulation platform involves interpreting the derived requirements for its detailed operation in the context of the virtual host's execution. The details are not given here and considered beyond the scope of this paper.

4. DEVS-Based M&S Capabilities and Tools for MBSE

Alshareef *et al.*[27] proposed an MBSE methodology that can support a full life cycle to design, model, and validate complex SoS. In this section, we review this methodology and relate it to the current litany of commercial MBSE tools as well as the DEVS-based artifacts in the research domain. Figure 2 places the methodology in relation to MBSE life cycle formulated to explicitly include M&S to support iterative stages of development ranging from system requirements to functional and system architectures as well as analysis and design optimization.

Beery[28] emphasized using systems modeling language products to link architecture and analysis. In contrast, the methodology proposed here advocates using the DEVS Formalism as the basic modeling and simulation framework for MBSE methodology to support the critical stages in the design of SoS. Included in Table 4 are columns outlining the development stages with examples from the application to emergency disaster response, commercial-off-the-Shelf (COTS) products that

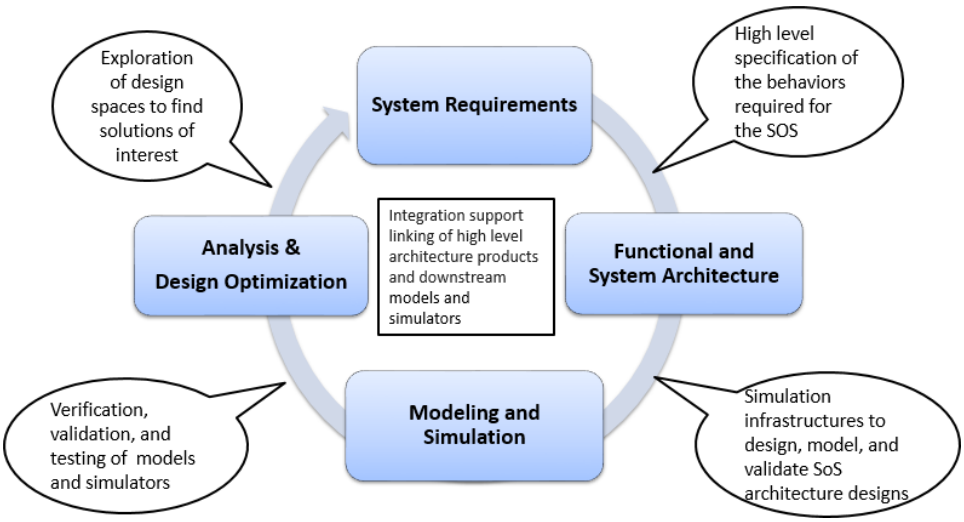

Fig. 2. MBSE life cycle formulated to explicitly include M&S to support the iterative stages.

Table 4. M&S support of the envisioned MBSE life cycle.

Development stage support	Example: Design of novel networks with capability to support emergency disaster response	COTS software	DEVS-based environments
High-level specification of the behaviors required for the SoS	High-level specification of packet routing in communication networks	MagicDraw+SysML Cameo Systems Modeler (NoMagic, 202)) IBM Rational/Raphsody (Mohlin, 2010)	UML/SysML metamodels that map to DEVS simulation models (Ref. 20, 34)
Simulation infrastructures to design, model, and validate SoS architecture designs	Models and simulators focusing on protocol evaluation, environmental representations in the context of hosted applications	SFI and Matlab/Simulink (MathWorks, 2018) OMNeT++[38]	CoSMoS (Component-based System Modeling and Simulation) (ACIMS, 2019) MS Me (MS4 Systems, 2018)
Integration support linking of high-level architecture products and downstream models and simulators	Virtual communication stack model provides framework for network modeling including one or more OSI layers at different resolution levels	Phoenix Model Center IEEE High-Level Architecture standard	HILLS (High-Level Language for System Specification)[37] Cadium DEVS[40]
Verification, validation, and testing of models and simulators	Hybrid and co-simulation approaches to packet traffic estimation offer complementary methods that together can overcome the limitations of each one	Cameo Simulation Toolkit (NoMagic, 2020)	Co-simulation of complex networks using DEVS as the formal basis, e.g., MECSYCO[36] Hybrid modeling of packet flow using PowerDEVS[35]
Exploration of design spaces to find the solutions of interest	Incorporation of intelligent search and learning methods to optimize routing parameters	Simulation-enabled multi-objective optimization[39]	DEVS-based parallel framework for multi-objective evolutionary algorithms[33]

support the listed functionalities, as well as current DEVS-based M&S tools and environments that are being developed in these areas. As detailed in the table, the approach starts with high-level specification of the behaviors required by the SoS to support system requirements engineering. This may be done using metamodels (e.g., UML/SysML) that map to simulation models (in formalisms such as DEVS) at an overall schematic level. Alshareef *et al.*[27] described the concepts and tools to support such specification and illustrated them with application to

high-level specification of packet routing mechanisms in communication networks. The application focused on evaluation of protocols, environmental representations, and included DEVS models of real network applications. DEVS-based simulation infrastructures to design, model, and validate SoS architecture designs were considered next and illustrated with DEVS models and simulators focusing on protocol evaluation, environmental representations in the context of hosted applications. For verification, validation, and testing of earlier developed models and simulators, two DEVS-based alternatives were discussed: (1) co-simulation of complex networks and (2) simulation-based testing using powerful hybrid fluid flow/packet-level mechanisms. The work showed that hybrid and co-simulation approaches based on DEVS offer complementary methods for the verification of SoS designs that together can overcome the limitations of each one. For integration support linking of high-level architecture products and downstream models and simulators, a virtual communication stack model was developed to provide a DEVS-based framework for network modeling that can flexibly include a selection of one or more standard Open Systems Interconnection (OSI) layers for abstraction at different levels of resolution. A DEVS-based framework for multi-objective evolutionary algorithms supports analysis and design optimization by exploration of design spaces to find solutions of interest using intelligent search and learning methods to optimize routing parameters.

4.1. *Mapping activity diagrams into executable activity-based DEVS models*

Alshareef *et al.*[29] discussed the integration of activity-based M&S into the MS4 Me environment demonstrating that support can be developed to link high-level architecture products and downstream models and simulators. Following practices of metamodeling, model transformation, and code generation, they implemented a capability to create UML activity diagrams and to map them into executable activity-based models in the DEVS Markov formalism.[30] This enabled formalization of the flow selection process and state transition in the activity diagram and parameterized specification of stochastic behavior. Such automated code generation supports fast development and simulation of activity-based models for high-level exploration of workflow architectures and illustration of system operation for customers and other stakeholders. Here we provide an example of such development.

Figure 3 illustrates an activity diagram with the graphical elements for input/output (Source/Store), splitting/merging of flow (Random_select/Merge), actions for work on jobs (Compare), and hierarchical construction (BaselineMultiProcessorArch/AlternatMultiProcessorArch) (see Ref. 29 for a more detailed description). At the top level, incoming jobs are randomly directed towards baseline and alternative workflow organizations with subsequent merging of these streams

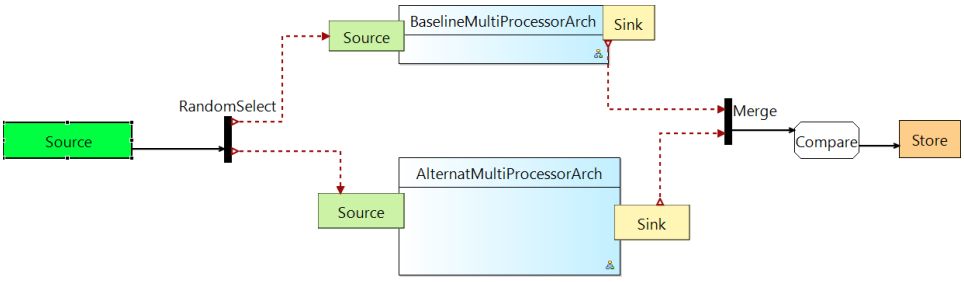

Fig. 3. Activity diagram determining the processing workflow based on random selection of jobs.

to enable comparison of their throughputs and turnaround times while minimizing the inference with ongoing production. The second-level elaboration of the two top-level interventions illustrates the capability of hierarchical construction to hide complexity at the top level while exposing it in consistent fashion at the next level down. Notice that although specified at the high level of activity modeling, the subsequent elaboration enables downstream implementation of this workflow.

However, the following limitations were noted:

- Mapped DEVS models in MS4 Me still require a multitude of parameter values to be set to achieve detailed desired behavior, although they are exposed to enable such adjustment.
- Changes in such models may put them outside the scope of activity-based specification, making "round-trip engineering" difficult.
- The current targeted messages and actions have to be enhanced to enable operations that generate the behavior required in other experimental frames. For example, currently the Compare action in Fig. 3 cannot actually perform comparison of intervention outcomes.
- The targeted atomic models cannot easily be used in compositions with other DEVS models.

4.2. *DEVS top-to-bottom MBSE capability*

To illustrate a full top-to-bottom MBSE capability, we proceed to show how the activity model developed in Fig. 3 can be manually integrated into full DEVS reusability. Figure 4(a) illustrates an SES that describes the coupled model structure underlying the activity diagram of Fig. 3. The baseline multi-processor architecture component is further represented in the SES of Fig. 4(b). Note that the same SES is employed for the alternative architecture (therefore not shown here) so that pruning can be done to select desired variants for both architectures from the same template. Roughly, the architecture consists of a coordinator and a group of processors that can implement single processor, multi-server, pipeline, and divide-and-conquer configurations (see Ref. 31 for discussion of such architectures). Figure 5 shows the Simulation Viewer display of an executable DEVS model that is generated

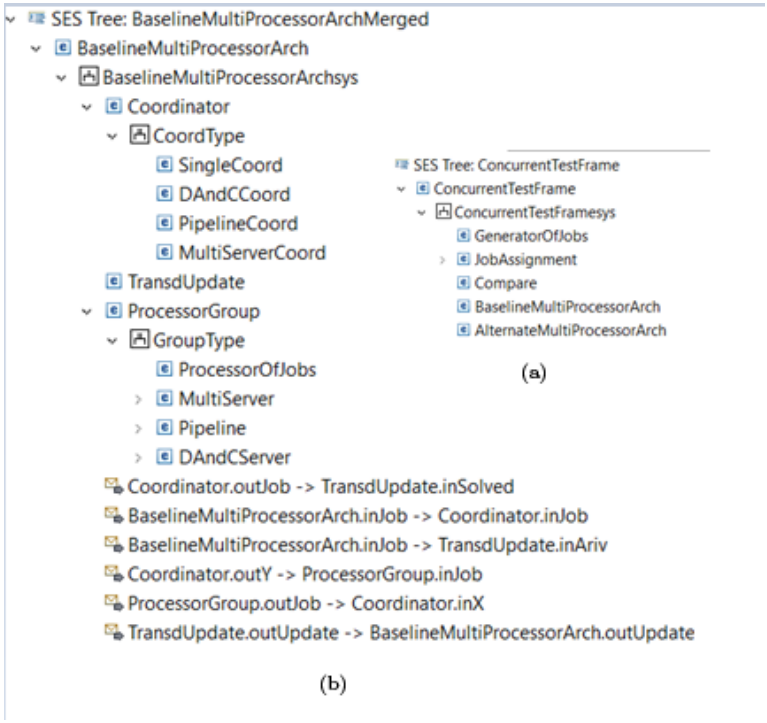

Fig. 4. SES that describes the coupled model structure underlying the activity diagram of Fig. 3.

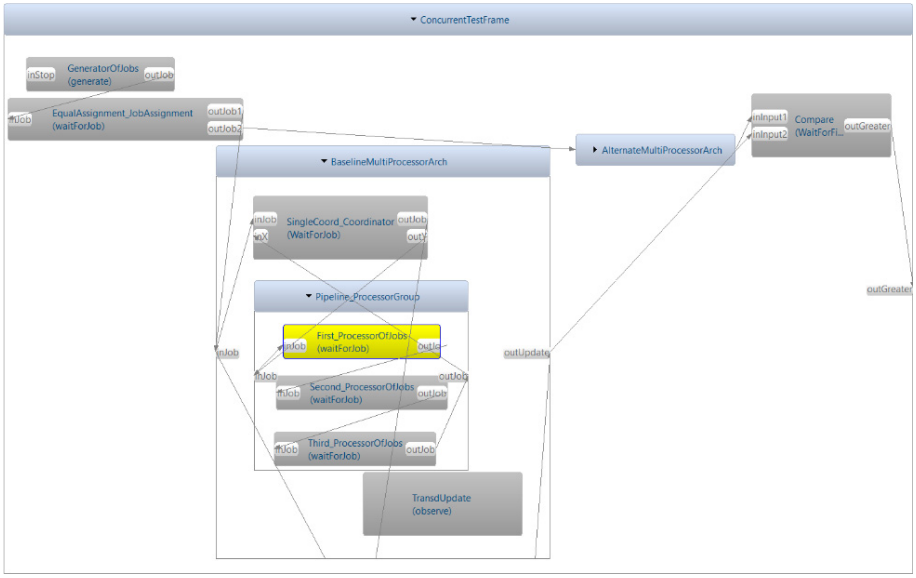

Fig. 5. Simulation Viewer display of a DEVS model pruned and transformed from the SES of Fig. 4.

by transforming such a pruning. The expanded baseline model shown in the Viewer corresponds to a choice of pipeline workflow model with three processors and its accompanying coordinator. The alternative, not expanded, is a divide-and-conquer group with accompanying coordinator.

Using capabilities identified in the last column of Table 4 automated pruning and transformation can be instrumented at this point to enable exploration of large design space in which baseline and alternative architectures are sampled for response to various workloads of interest. Thus, to fully support a top-to-bottom design process initiated by the activity diagram would require integration in MS4 Me that also supports semi-automated development of SES and DNL files that then merge with those of other DEVS models. Research is needed to understand whether and how activity diagram specification can be extended to enable such generation of SES and DNL documents to increase model reusability.

5. Summary and Conclusions

Having reviewed DEVS in the context of MBSE, we discussed the application of DEVS methodology to MBSE. We discussed homomorphic implementation of DEVS-like systems including the hierarchy of system specification morphisms and an application to the design of simulation systems for "as-is" software. We then focused on DEVS-based capabilities and tools for MBSE with discussion of support of an envisioned MBSE life cycle with DEVS top-to-bottom MBSE capability. As an example, we discussed a mapping of activity diagrams into executable activity-based DEVS models implemented in the MS4 Me environment. The result was further elaborated into a family of architectural variants using the System Entity Structure. Integrating the activity modeling with the elaborations supported by the latter concepts and tools remains for continued development to achieve.

References

1. Wymore W., *A Mathematical Theory of Systems Engineering: The Elements*, Wiley, New York, 1967.
2. Zeigler B. P., Muzy A., Kofman E., *Theory of Modeling and Simulation,* 3rd edn., Academic Press, New York, NY, 2018.
3. Seo C., Zeigler B. P., Coop R., Kim D., DEVS modeling and simulation methodology with MS4 Me software tool, *DEVS 13: Proc. Symp. Theory of Modeling & Simulation — DEVS Integrative M&S Symp.*, San Diego, CA, The Society for Modeling & Simulation International, San Diego, CA, pp. 33:1–33:8, 2018.
4. MS4 Systems, Inc., MS4 Me, http://ms4systems.com/pages/ms4me.php, 2021.
5. Zeigler B. P., Sarjoughian H. S., *Guide to Modeling and Simulation of Systems of Systems*, 2nd edn., Simulation Foundations, Methods and Applications, Springer International Publishing, London, 2017.
6. Zeigler B. P., Nutaro J., Towards a framework for more robust validation and verification of simulation models for systems of systems, *J. Def. Model. Simul., Appl. Methodol. Technol.* **13**:3–16, 2015, doi:10.1177/1548512914568657.

7. Zeigler B. P., *Multifaceted Modelling and Discrete Event Simulation*, Academic Press Professional, Inc, San Diego, CA, 1984.
8. Paredis, R., Van Mierlo, S., Vangheluwe, H., Translating Process Interaction World View Models to DEVS: GPSS to (Python(P))DEVS, 2020 Winter Simulation Conference (WSC), 2020, pp. 2221–2232, doi: 10.1109/WSC48552.2020.9383952.
9. Zeigler B. P., Nutaro J., Seo C., What's the best possible speedup achievable in distributed simulation: Amdahl's law reconstructed, Proc. 2015 Spring Simulation Multi-Conf., Alexandria, VA, The Society for Modeling & Simulation International, San Diego, CA, pp. 189–196, 2015.
10. Wainer G. A., DEVS tools, http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm, 2012.
11. Franceschini R., Bisgambiglia P.-A., Touraille L., Bisgambiglia P., Hill D., A survey of modelling and simulation software frameworks using discrete event system specification, in Neykova R., Ng N. (Eds.), *2014 Imperial College Computing Student Workshop*, OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl — Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 40–49, 2014.
12. Van Tendeloo Y., Vangheluwe H., An evaluation of DEVS simulation tools, *SIMULATION* **93**:103–121, 2017.
13. Nutaro J., On constructing optimistic simulation algorithms for the discrete event system specification, *ACM Trans. Model. Comput. Simul.* **19**:1:1–1:21, 2008.
14. Nutaro J., *Building Software for Simulation: Theory and Algorithms with Applications in C++*, John Wiley & Sons, Hoboken, NJ, 2011.
15. Vangheluwe H., Foundations of modelling and simulation of complex systems, *Electron. Commun. EASST* **10**:1–12. 2008.
16. Zeigler B. P., *Theory of Modeling and Simulation*, Wiley-Interscience, New York, 1976.
17. Chow A., Parallel DEVS: A parallel, hierarchical modular modeling formalism and its distributed simulator, *SIMULATION, Trans. Soc. Comput. Simul. Int.* **13**:55–68, 1996.
18. Zeigler B. P., Hierarchical, modular discrete event models in an object oriented environment, *SIMULATION* **49**:219–230, 1987.
19. Kim T. G., Lee C., Zeigler B. P., Christensen E. R., System entity structuring and model base management, *IEEE Trans. Syst. Man Cybern.* **20**:1013–1024, 1990.
20. Barros F. J., Dynamic structure discrete event system specification: structural inheritance in the delta environment, *Winter Simulation Conference Proceedings*, pp. 781–785, 1995.
21. Praehofer H., System theoretic formalisms for combined discrete-continuous system simulation, *Int. J. Gen. Syst.* **19**:226–240, 1991.
22. Kofman E., Junco S., Quantized-state systems: A DEVS approach for continuous system simulation, *SIMULATION*, *Trans. Soc. Model. Simul. Int.* **18**:2–8, 2001.
23. Giambiasi N., Escude B., Ghosh S., GDEVS: a generalized discrete event specification for accurate modeling of dynamic systems, *Proc. 5th Int. Symp. Autonomous Decentralized Systems*, Dallas, TX, IEEE, Piscataway, pp. 464–469, 2001.
24. Nutaro J., An extension of the OpenModelica compiler for using Modelica models in a discrete event simulation, *SIMULATION* **90**:1328–1345, 2014.
25. Hwang M. H., Taxonomy of DEVS subclasses for standardization, *Proc. 2011 Symp. Theory of Modeling & Simulation: DEVS Integrative M&S Symp.*, Boston, MA, The Society for Modeling & Simulation International, San Diego, CA, pp. 152–159, 2011.
26. Wach P., Zeigler B. P., Salado A., Conjoining Wymore's systems theoretic framework and the DEVS modeling formalism: Toward scientific foundations for MBSE, *Appl. Sci.* **11**:4936, 2021.

27. Alshareef A., Blas M., Bonaventura M., Paris T., Zeigler B. P., Using DEVS for full life cycle, in *Model-Based System Engineering in Complex Network Design*, Springer NY, New York, 2021 (in press).
28. Beery P., A model based systems engineering methodology for employing architecture in system analysis: Developing simulation models using systems modeling language products to link architecture and analysis, Master's Thesis, Naval Postgraduate School, Monterey, CA, 2018.
29. Alshareef A., Kim D., Seo C., Zeigler B. P., DEVS Markov modeling and simulation of activity-based models for MBSE application, *Proc. Winter Simulation Conf. 2021*, Phoenix, AZ, 2021.
30. Seo C., Zeigler B. P., Kim D., DEVS Markov modeling and simulation: Formal definition and implementation, *Proc. Theory of Modeling and Simulation Symp.*, Baltimore, MA, The Society for Modeling & Simulation International, San Diego, CA, pp. 1–12, 2018.
31. Alshareef A., Sarjoughian H. S., Hierarchical activity-based models for control flows in parallel discrete event system specification simulation models, *IEEE Access* **9**:80970–80985, 2021, doi:10.1109/ACCESS.2021.3084940.
32. Alshareef, A., Kim, D., Seo, C., Zeigler, B. P., Activity diagrams between DEVS-based modeling and simulation and fUML-based model execution, in *Proc. 2020 Summer Simulation Conf.*, Society for Computer Simulation International, 2020.
33. Astorga, A. M. *et al.*, DEVS-based parallel framework for multi-objective evolutionary algorithms, in *The Fourth Int. Workshop on Parallel Architectures and Bioinspired Algorithms — The Twentieth Int. Conf. Parallel Architectures and Compilation Techniques*, 2011.
34. Blas, M. J., Gonnet, S., Leone, H., Routing structure over discrete event system specification: A DEVS adaptation to develop smart routing in simulation models, Winter Simulation Conf. (WSC), IEEE, pp. 774–785, 2017.
35. Bonaventura, M., Castro, R., Fluid-flow and packet-level models of data networks unified under a modular/hierarchical framework: Speedups and simplicity, combined, *Proc. of 2018 Winter Simulation Conf.* (WSC), 2018.
36. Camus, B., Paris, T., Vaubourg, J., Presse, Y., Bourjot, C., Ciarletta, L., Chevrier, V., Cosimulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware. SIMULATION, 2018, doi:10.1177/0037549717749014.
37. Kehinde, G., and Mamadou, K. T., A DEVS-based pivotal modeling formalism and its verification and validation framework, *Simulation J.* **96**(12) (2020) 969–992.
38. Varga, A., Hornig, R., An overview of the OMNeT++ simulation environment, *Proc. 1st International Conf. Simulation Tools and Techniques for Communications*, Networks and Systems & Workshops, Simutools'08, ICST, Brussels, Belgium, pp. 60:1–60:10, 2008, http://dl.acm.org/citation.cfm?id=1416222.1416290, http://dl.acm.org/citation.cfm?id=.
39. Wade, B. M., A multi-objective optimization of ballistic and cruise missile fire plans based on damage calculations from missile impacts on an airfield defended by an air defense artillery network, *J. Defense Model. Simul.*, 1–15, 2018, doi: 10.1177/1548512918788503.
40. Wainer, G. A., http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm.