Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

EXTENSIBLE DISCRETE-EVENT SIMULATION FRAMEWORK IN SIMEVENTS

Wei Li
Ramamurthy Mani
Pieter J. Mosterman

MathWorks
1 Apple Hill Dr.
Natick, MA 01760, USA

ABSTRACT

A simulation framework is introduced that facilitates hierarchical definition and composition of discrete-event systems. This framework enables modelers to flexibly use graphical block diagrams, state charts, and MATLAB textual object-oriented programming to author custom domain-specific discrete-event systems. The framework has been realized in an implementation that spans multiple software simulation tools including SimEvents, Stateflow, Simulink and MATLAB.

1 INTRODUCTION

Over the past several years, an important development has been taking place in the field of embedded control systems. Previously, the implementation of digital control would rely on microcontroller architectures that were uniform in their single threaded and instruction set nature. More recently, however, much more heterogeneous target architectures are being exploited. Multi-core processors are now combined with graphics processing units (GPU) and field programmable gate arrays (FPGA) in an attempt to keep pace with the rapidly growing needs for computationally intensive applications. For example, architectures to support self-driving cars require not only traditional low latency and high performance feedback control loops but also high throughput video processing functionality, which may best be implemented on a GPU or FPGA.

In addition to the variability in target architecture, another recent development stems from the increasing connectivity of systems (e.g., 'things' in the Internet of Things). For example, vehicles may communicate with each other to share improvements in functionality thanks to machine learning techniques. Moreover, systems may configure into system ensembles to collaborate on tasks such as emergency response (Mosterman and Zander 2016; Mosterman et al. 2014).

Both of these developments have put forward an urgent need for powerful and sophisticated discrete event modeling methods because they are uniquely suited to characterize target architecture performance and system-level properties. Because of the increasing variability in architectures and system configurations these characterizations are essential in design feasibility as well as in trade studies. Importantly, such characterizations and related models have risen to prominence with the abandonment of the formerly more uniform and fixed forms of architectures and configurations.

The discrete event modeling methods must also integrate with other models of computation such as discrete time to model sampled control behavior, continuous time to model physical behavior, state transition logic to model supervisory control, sequence control, etc., and imperative algorithmic computation to model signal and image processing, convolutional neural network learning, performance optimization, etc. An integrated framework would be essential to facilitate design and analysis of highly complex systems that are coming online such as fleets of self-driving cars, swarms of robots, airspace

sharing autonomous and manned vehicles, highly interconnected power grid nodes, etc. (Sztipanovits et al. 2013). The complexity of these systems mandates abstraction to efficiently and effectively address the very heterogeneous functionality they embody and integrate (e.g., computer vision algorithms, feedback control laws, optimization programs, machine learning networks, supervisory plans, and batch control operations).

Another trend is the emerging Industry 4.0 paradigm (Industrie 4.0 Working Group 2013) that integrates enterprise level information flows with manufacturing cell operations to achieve optimized manufacturing plant operation across the various operational layers of the entire enterprise. Such optimization may rely on a generative approach that defines a flexible and powerful interface between the problem space (the optimization problem) and the solution space (the manufacturing plant operations). This interface may be in the form of a Knowledge Interface Broker (KIB) to combine discrete event simulation with model predictive control (MPC) optimization algorithms that operate on a much larger time scale (Godding, Sarjoughian, and Kempf 2007; Huang et al. 2009). The value of an integrated architecture that refines the KIB has been shown to enable solving generative optimization problems for resources allocation (Li, Cassandras, and Clune 2006; Hubscher-Younger et al. 2012). This exploits the potential of discrete-event modeling and simulation technology that is integrated with other models of computation.

In 2006, Mathworks introduced SimEvents® (MathWorks 2016c) as a tool that adds Discrete-Event Simulation (DES) capabilities to the Simulink® (MathWorks 2016d) modeling platform. Clune, Mosterman, and Cassandras (2006) illustrated how the product was equipped to mix the multidomain time-driven computation of Simulink with event-driven simulation for modeling complex hybrid systems. Since that time, SimEvents has been successfully applied to simulate a variety of application scenarios including modeling of real-time operating systems, communication networks, operations research problems in manufacturing and logistics, and hardware modeling. However, analysis of the product and comparison to existing solutions (Gray 2007) have identified certain desirable characteristics that are not present in the tool including the lack of a framework for customization and the lack of a published formalism (Schwatinski et al. 2010; Seo et al. 2014).

This paper presents the new Discrete-Event Simulation framework that was introduced in SimEvents by MathWorks (2016). The new framework overcomes these shortcomings and is in a position to serve as a foundation upon which modelers can easily build application and domain-specific systems. By drawing on the formal system engineering research effort by Wymore (1993), the aim is to provide a framework that is as powerful and versatile as other similarly inspired efforts such as the DEVS formalism (Zeigler 1984; Concepcion and Zeigler 1988; Wainer 2009).

Thanks to this new SimEvents framework, modelers have access to the combined use of multiple graphical and textual modeling languages to create highly customized discrete-event systems. These modeling languages include the graphical, queueing-system based language of SimEvents, the graphical, finite-state machine based language of Stateflow® (MathWorks 2016e), and the textual, objected-oriented programming language of MATLAB® (MathWorks 2016b). In conjunction with the discrete-event/continuous-time hybrid simulation engine of the original SimEvents (Clune, Mosterman, and Cassandras 2006), a single simulation model can include both discrete-event components implemented by all of the above three languages, and continuous-time components implemented by Simulink.

Section 2 describes the modeling formalism that the proposed simulation framework establishes. Section 3 discusses the simulation framework including common modeling primitives and the differing language representations in SimEvents, Stateflow, and MATLAB. Section 4 gives an application example to demonstrate how these multi-domain and hybrid simulation capabilities can help the design of today's cyber-physical system. Section 5 summarizes and presents a look to the future.

2 MODELING FORMALISM

Previous versions of SimEvents predating 2016 relied on a simulation engine that performed simulation through run-time handshaking between atomic DES components. The implication of this is that the

engine did not compile a network of interconnected components into a composite system description before simulating the abstract composition. As a result of not relying on a composable formalism to describe a DES system, these SimEvents versions did not publish a system definition upon which modelers could define their specialized domain-specific components. This directly caused a lack of customization observed by Gray (2007). To overcome these issues, SimEvents (MathWorks 2016c) defines a new formalism inspired by the systems theoretic foundations research of Wymore (1993) that describes a discrete system as the quintuple:

$$Z = (S, I, O, N, R)$$

where:

Z is the $name$ of the system,
S is the set of $states, S \neq \emptyset$,
I is the set of $inputs$,
O is the set of $outputs$,
N is the $next\ state\ function, N \in FNS(S \times I, S)$ if I is not empty, otherwise $N \in (S, S)$,
R is the $readout\ function, R \in FNS(S, O)$ if O is not empty, otherwise $R = \emptyset$.

Within this new formalism, termed SimEvents Entity-Storage (SEES) formalism, an abstraction termed 'entities' is introduced as packets of data whose handling and transport in the discrete-event system produces events. Additionally, the formalism provides a built-in primitive abstraction of a discrete-event component called 'storage' (for entities) which maps to queues and servers that are common in various flow-based DES simulation packages. Storages are effectively containers for entities that may be defined as follows:

$$I = \begin{Bmatrix} (\text{generate, entity}), (\text{destroy, entity}), (\text{entry, entity}), (\text{exit, entity}), \\ (\text{blocked, entity}), (\text{timer, entity}), (\text{iterate}) \end{Bmatrix}$$

$$O = \begin{Bmatrix} (\text{schedule\_generate}), (\text{schedule\_destroy, entity}), (\text{schedule\_exit,entity}), \\ (\text{schedule\_timer,entity}), (\text{schedule\_iterate}) \end{Bmatrix}$$

$$S = \{\text{occupancy} \in \{0,1\}^*, \text{storage} \in \textbf{entity*}, \text{custom state}\}$$

Here, I and O are sets of input and output events that correspond to a storage location. Additionally, in SEES, five specific functions are recognized that advance the state of the system and also produce outputs based upon the occurrence of five classes of input events: (1) Generate: creation of a new entity, (2) Entry: entry of an input entity, (3) Iterate: iteration over the contents of the storage, (4) Blocked: failure of an output event for entity from this storage, and (5) Timer: expiry of a timer. Also, the SEES component has two additional functions that may advance the state of the system but not produce any output based on the occurrence of the following events: (1) Exit: exit of an entity from that storage and (2) Destroy: destruction of an entity.

Coupled SEES components are natural in the Simulink environment where components are represented by blocks that have a port based interface and where directed entity connections are established between ports. The core architecture implementing SEES uses the connections to analyze and synthesize the coupled formalism prior to commencing the simulation. The simulation then relies on simulating the coupled model directly. In addition, entities are associated with the notion of priority which helps act as the basis for implementing a tie-breaker function. The management of the complex temporal events of the SEES system are arbitrated by an event calendar that relies on this tie-breaker function.

Note that the coupling between atomic components in SEES is similar to the notion of coupled DEVS (Zeigler 1984). Like SEES, a coupled DEVS model introduces extra concepts of (1) external input couplings, external output couplings, and internal couplings to capture connections between atomic DEVS components, and (2) a tie-breaker function to determine which atomic DEVS in a composition takes precedence during a collision of events.

Providing support in SEES for atomic and coupled discrete-event components has the potential to enable modelers to leverage various powerful analysis and verification techniques that have emerged in research (Holzmann 2004). Additionally, by building the abstraction of entity and storage, the framework aims to help very quickly compose complex flow-based systems common in DES applications. The next section discusses how such compositions may be created using graphical block diagrams, state charts, and textual programming all working in union. As demonstrated in the next two sections, these abstractions allow modelers to quickly express their discrete-event system as interconnections of storages interacting via entities while also being firmly grounded in the underlying formalism.

3 MODELING FRAMEWORK

The introduced formalism is realized within MATLAB and Simulink as a framework that includes various graphical and textual programming paradigms. This variety of programming choices provides modelers with the ability to choose the paradigm that is appropriate for the modeling context. Each of the paradigms and the notion of entity is described in the remainder of this section.

3.1 Entity

An entity consists of structured data (called *attributes*) that is bundled as payload of the events at the inputs and outputs of DES systems in SEES. By providing such a general notion of data, one can associate a rich set of domain-specific data with an event. For example, in a communication modeling context, an entity could encapsulate a communication message whose data fields include attributes related to the message's destination address, encoding, and payload. SEES components that model the communication channel would accept and produce events that correspond to the arrival and departure of such messages. Additionally, when the entity is within a storage, its attributes implicitly become part of the state of that storage. As will be illustrated in the remainder of this section, one can have access to the data/attributes in each of the programming paradigms of SEES with the syntax of a MATLAB structure datatype.

Entity lines and ports define interfaces of SEES components for data sharing and event triggering. Connecting two SEES components via an entity line establishes a route for instantaneous transfer of entities accompanied by triggering of the input and output events defined in the previous section. In addition, SEES components can also share their internal state by using Simulink signal lines and ports. Similar to semantics of Simulink, a signal line is simply a persistent value that has no implications on event triggering. For example, in the model of Figure 6, three SEES blocks transfer entities using entity lines (i.e., double-line style). In addition, the Entity Terminator block produces a signal as output (i.e., single-line style) for the purpose of plotting its internal state corresponding to number of entities that have arrived at the Terminator.

3.2 Graphical Programming

SEES introduces a set of basic blocks that can be used to compose a discrete-event system from primitive components. These blocks, which are each built from one 'storage' component introduced in the previous section, include Generators, Queues, Servers, and Sinks. In addition, there are blocks for routing such as Switches and Gates that only influence the coupling functions of a DES composition and are not atomic discrete-event components themselves. Advanced functions such as replication, combining, and batching are also included that are themselves atomic or coupled discrete-event components. Finally, the graphical blocks may also be connected directly to time-driven components of Simulink to integrate DES with time-driven simulations as described by Clune, Mosterman, and Cassandras (2006).

The graphical programming paradigm derives its extensibility from allowing modelers to specify the various functions corresponding to the input events described in the previous section for the underlying storage discrete-event component. Each of these functions is represented as a specialized 'event action' that is associated with that specific class of event. For example, the Entity Generator block of the M/M/1 queueing system (Cassandras 2009) model shown in Figure 1 has a 'Generate' action as shown by Figure

2. This custom code illustrates how one of the attributes of the entity is being initialized by a random number in the 'Generate' action associated with the 'Generate' event. The action is expressed in the familiar MATLAB language syntax with the data/attributes of an event/entity being accessed as MATLAB structures. One can similarly access the other functions (such as Entry and Exit functions) in a user-interface dialog for the Queue and Server blocks in this model.

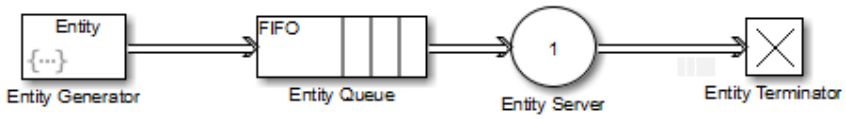

Figure 1: A M/M/1 queueing system model with custom state-transition functions.

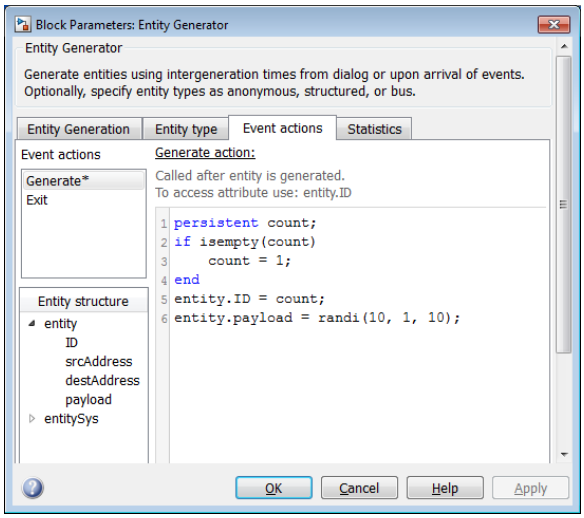

Figure 2: Custom event actions of Entity Generator block.

3.3 State Charts

SEES also extends seamlessly to the state chart paradigm (Harel 1987) supported by Stateflow, a tool for modeling state transition behavior. A new type of state chart has been introduced called DES chart that has entity ports (referred also as 'message' ports in the GUI). The chart itself may be an atomic discrete-event component as illustrated below where the chart is used to implement the server in an elementary M/M/1 system. The notion of SEES storage discussed in the previous section underlies every entity port in Stateflow. The state-transition language may then be used to express the functions described in Section 2 with a combination of previously available Stateflow syntax.

As illustrated in the example implementation of the Server of Figure 3 in Figure 4, the state transition functions are directly annotated on the transition actions from one state to the next. When Server is in 'Idle' state, upon entry of an entity at the 'Input' port, Server transits its state to 'Busy'. As a part of the transition action, the 'forward' keyword is used to declare an event that forwards the incoming entity to an internal storage (named 'Job') of the chart. Additionally, the 'after' keyword is used to scheduler a Timer event as well as Timer Function. A detailed description of the use of this notation is documented in the SimEvents User's Guide (MathWorks 2016c).

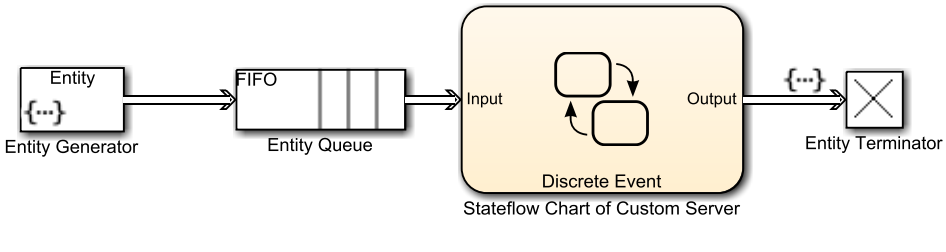

Figure 3: Example custom event actions of Entity Generator block and Entity Sever block.

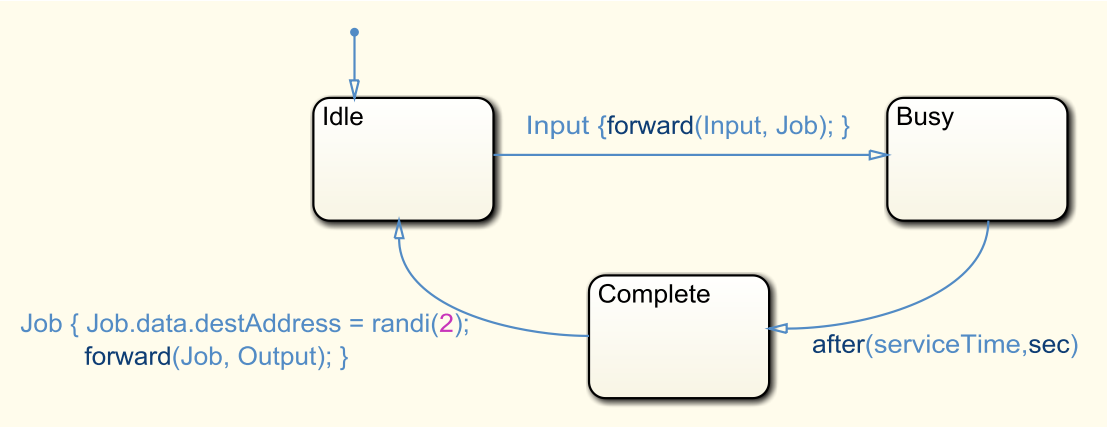

Figure 4: Example custom server modeled using a discrete-event Stateflow chart (DES chart).

In addition to expressing atomic discrete-event components as state charts, one can also compose a coupled discrete-event component directly in Stateflow. This involves either declaring additional event ports or internal 'storage' within the chart. The SEES architecture once again is able to analyze and synthesize the coupled discrete-event model before DES simulation is performed.

3.4 Textual Programming

For maximal flexibility modelers can program arbitrarily complex discrete-event systems using a MATLAB class-based program. A MATLAB class extension that realizes SEES components has been developed as the *MATLAB Discrete-Event System (MDES)*. A critical advantage of this approach is that the class can interoperate with all MATLAB functions and toolboxes. Additionally, the ability to programmatically define arbitrary compositions of atomic storage components offers a platform for quickly defining large networks such as a wireless computer network or sensor network.

The MDES extension is in spirit akin to the multi-language S-Function API (MathWorks 2016f) of the Simulink platform and allows the authoring of arbitrary systems in the discrete-time model of computation. In addition, the MDES extension directly builds upon the MATLAB System Object notation (MathWorks 2016a), and is supported by the Simulink platform. In this manner, arbitrary systems can be represented in a discrete-time model of computation using the MATLAB language. While reusing features of the notation such as the definition of input and output (ports), component parameters, and component states, the new extension adds the ability to define entity input and output ports, and an arbitrary number of atomic discrete-event storage components. Modelers can then couple these atomic components using specialized notation that defines their connectivity. Finally, modelers can also once

again define custom code for the functions defined in Section 2. A full definition of the API is provided in the SimEvents User's Guide (MathWorks 2016c).

Figure 5 shows a simple example of defining a server block as in a *M*/*D*/*n* queueing system (Cassandras 2009). The program specifies component parameters as well as the definition of a single storage that is already pre-configured as a FIFO queue within the getEntityStorageImpl method of the class. In addition, code for 'Entry' function that handles the 'entry' event is shown. The line with 'eventForward' then registers both a 'timer' event and an additional 'exit' event to forward the corresponding entity to the output port of the block upon Timer expiry. As illustrated in Figure 6, the component defined in Figure 5 can now be simply imported as a block into Simulink. SEES then uses analysis to compose this block with other interconnected blocks to perform a full DES simulation.

```
classdef myServer < matlab.DiscreteEventSystem
    % Custom server with capacity and service time configured as parameters
    properties (Nontunable)
        % Number of servers (nontunable parameter)
        Capacity = 5;
    end
    properties
        % Service time (tunable parameter)
        ServiceTime = 1.0;
    end
    methods (Access = protected)
        function [storageSpec, I, O] = getEntityStorageImpl(obj)
            % Create storage with capacity from parameter value
            storageSpec = obj.queueFIFO('entity', obj.Capacity);
            I = 1; % Connect storage to input port
            O = 1; % Connect storage to output port
        end
        function [entity, events] = entryImpl(obj, storage, entity, from)
            % Forward entity to output port after service completes
            events = obj.eventForward('output', 1, obj.ServiceTime);
        end
    end
end
```

Figure 5: Example custom server defined using a MATLAB Discrete-Event System.

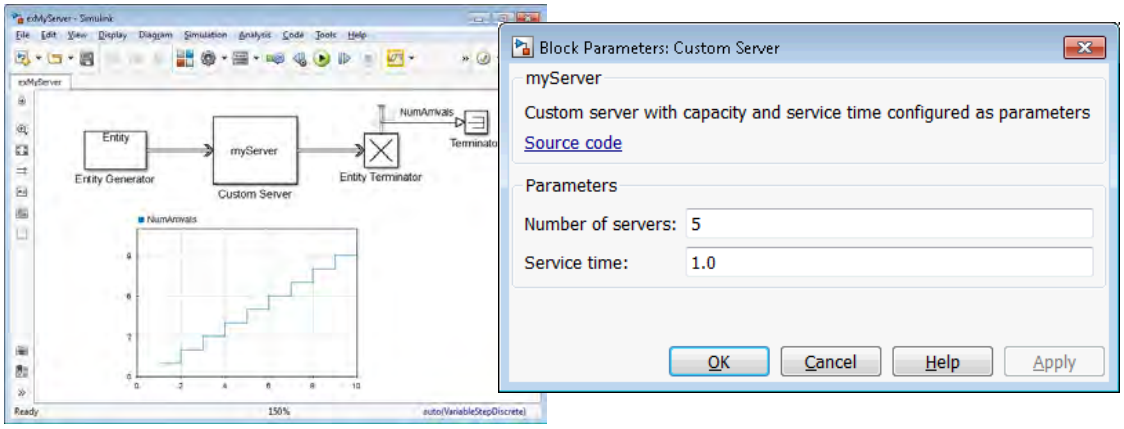

Figure 6: Example of a MATLAB® Discrete-Event System block and its GUI dialog.

4 APPLICATION EXAMPLE

As discussed in Section 1, simulation of today's complex multicore and distributed systems requires the combined use of multiple modeling paradigms and hybrid simulation technologies with discrete-event simulation. The ability to perform such simulations in SEES is illustrated with the example shown in Figure 7. The model in Figure 7 represents a prototype robotic system (modeled as a cart with an inverted pendulum) that has a camera mounted on it. The system receives commands for the cart to go to a new position along the horizontal x-axis periodically. The primary high speed control loop with a sampling time of 10 msec then attempts to reach this new position while keeping the robot (i.e., the pendulum) balanced. Additionally, the robot also receives periodic requests for setting a new camera position. This is achieved using a slow control loop with a sampling time of 1 sec. Sporadic health monitoring requests are received approximately every 10 msec. To study practical feasibility of deploying this system in real-time hardware, the effects of running this prototype system on a 'virtual' real-time operating system that supports preemption are emulated and a comparison is made between system performance with a single-core and two-core architecture.

The model in Figure 7 consists of:

- **Simulink continuous-time components** of the motion dynamics of the pendulum system and the camera labeled 'Plant'.
- **Simulink discrete-time runnable** modeled using Simulink function-call subsystem notation labeled as 'Control Algorithms' that model the two Control loops and Health monitoring components.
- **Discrete-event Stateflow chart** in the 'Plant' section that specifies the power-up sequence of the pendulum system and its embedded controller, as well as the generation of the health monitoring requests.
- **MATLAB discrete-event System block and SimEvents Server block** that model the real-time operating system shown in the 'Real-Time Environment Emulation' section. These blocks rely on a 'Runnable Broker' to dispatch the Simulink control algorithm computation based on timing from the operating system (OS) emulation component. The OS Emulator block is implemented with MATLAB Discrete-Event System technology and models a pre-emptible multicore operating system using a few hundred lines of MATLAB code. The parameters of this block are captured using the GUI dialog shown in Figure 8. This GUI helps configure the computational latencies of each runnable that the model will attempt to emulate. For example, Runnable 'Cont1Ctrl' corresponding to the core pendulum controller is expected to require 1.5 msec +/- 0.5 msec for every execution. It is assumed that the modeler obtained this number either as a design budget or from estimation on the actual hardware.

When the simulation is performed while emulating a single core system, the yellow plot in Figure 9 is obtained for the position error of the pendulum. One can clearly see the effect of jitter caused in scheduling because of pre-emption by the health monitoring task of the pendulum controller. If the number of cores is changed from 1 to 2 simply by modifying the value on the GUI dialog of the OS emulator block, one sees that the addition of a dedicated core for handling the health monitoring interrupt eliminates the jitter in the green plot of Figure 9.

This example conveys the power of being able to integrate discrete-event simulation into a classical control problem in the context of emerging needs for studying the effect of multicore scheduling. Additionally, it also illustrated how such a system can leverage the capabilities of a single framework that flexibly supports multiple paradigms to express DES components.

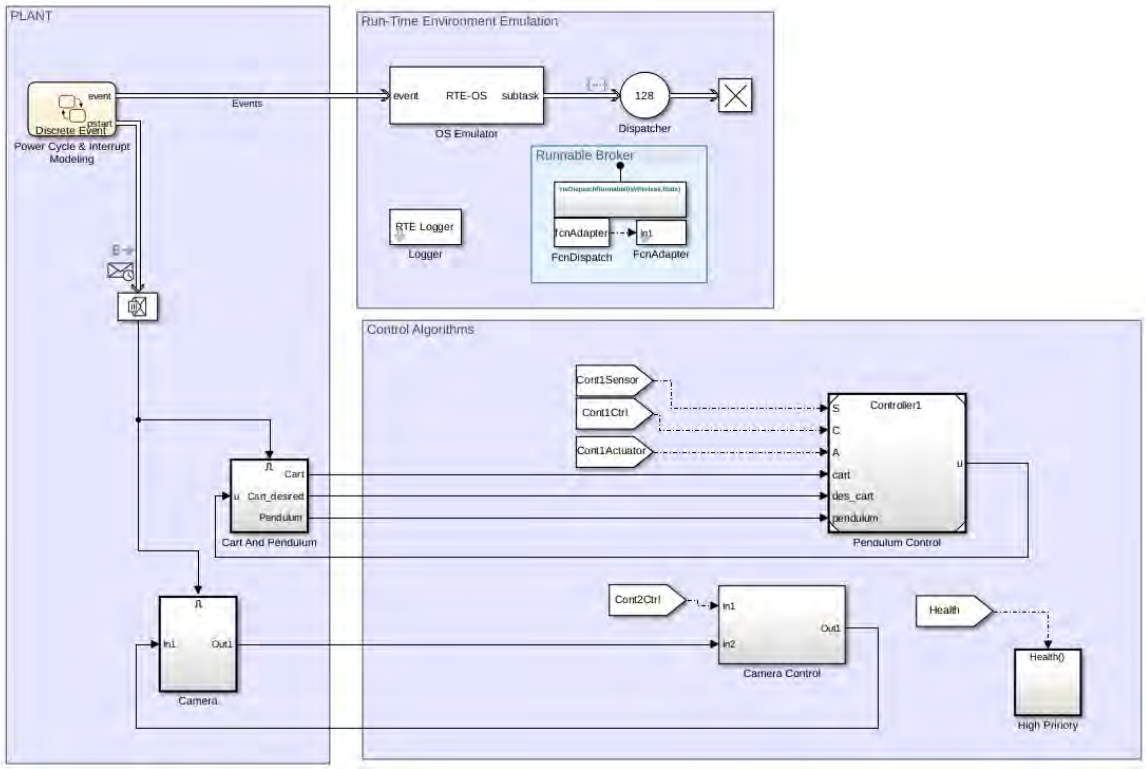

Figure 7: Application example of real-time operating system.

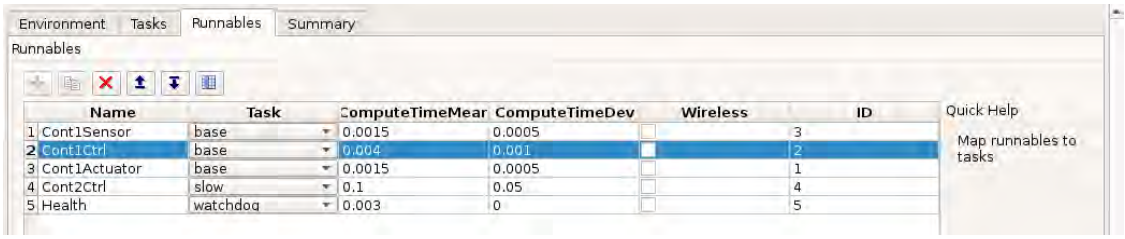

Figure 8: Configuring the amount of time each Runnable requires (i.e., computational latency).

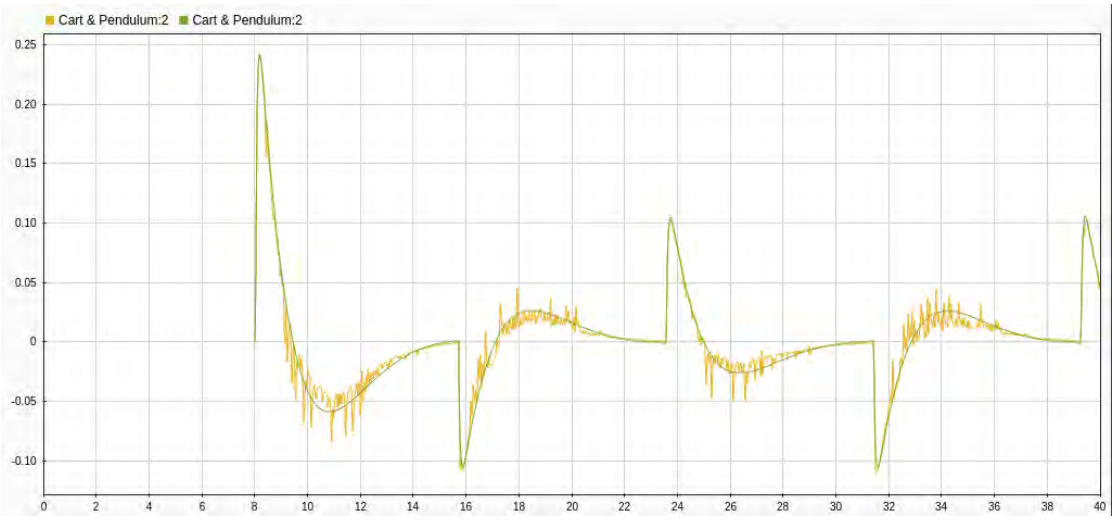

Figure 9: Results of running the simulation that show deviation of the pendulum from vertical. The yellow line shows the pendulum position that is influenced by scheduling jitter on a single-core system. The green line shows the elimination of jitter by using a two core system.

5 SUMMARY AND NEXT STEPS

A new framework for Discrete-Event System (DES) modeling and simulation is introduced that is built upon the existing platform provided by MATLAB and Simulink. The framework introduces abstractions that help easily express and compose DES systems in various graphical and textual programming paradigms. The framework draws inspiration from system theoretic work by Wymore (1993) which aims to position it well for future work on analysis and verification techniques. The well founded SimEvents Entity-Storage formalism (SEES) intends to allow modelers to efficiently design and reason about systems built in this framework. In future work, exploring a formal relationship between SEES and DEVS will help bring a well-studied theoretical basis to SEES and provide possible new avenues of research for DEVS.

REFERENCES

Industrie 4.0 Working Group. 2013. *Securing the Future of German Manufacturing Industry. Recommendations for Implementing the Strategic Initiative*. National Academy of Science and Engineering, Munich.

Cassandras, C. G., and S. Lafortune. 2009. *Introduction to Discrete Event Systems*. Springer Science & Business Media.

Clune, M. I., P. J. Mosterman, and C. G. Cassandras. 2006. "Discrete Event and Hybrid System Simulation with SimEvents". In *Proceedings of the 8th International Workshop on Discrete Event Systems*: 386-387.

Concepcion, A. I., and B. F. Zeigler. 1988. "DEVS Formalism: A Framework for Hierarchical Model Development". *IEEE Transactions on Software Engineering* 14(2): 228.

Godding, G., H. Sarjoughian, and K. Kempf. 2007. "Application of Combined Discrete-Event Simulation and Optimization Models in Semiconductor Enterprise Manufacturing Systems". In *Proceedings of the 2007 Winter Simulation Conference*. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.: 1729-1736. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gray, M. A. 2007. "Discrete Event Simulation: A Review of SimEvents". *Computing in Science & Engineering* 9(6): 62-66.

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

Holzmann, G. J. 2004. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley.

Huang, D., H. S. Sarjoughian, W. Wang, G. Godding, D. E. Rivera, K. G. Kempf, and H. Mittelmann. 2009. "Simulation of Semiconductor Manufacturing Supply-Chain Systems with DEVS, MPC, and KIB". *IEEE Transactions on Semiconductor Manufacturing* 22(1).

Li, W., C. G. Cassandras, and M. Clune. 2006. "Model-Based Design of a Dynamic Voltage Scaling Controller Based on Online Gradient Estimation Using SimEvents." In *Proceedings of the 45th IEEE Conference on Decision & Control*, San Diego, California.

Hubscher-Younger, T., P. J. Mosterman, S. DeLand, O. Orqueda, and D. Eastman. 2012. "Integrating Discrete-Event and Time-Based Models with Optimization for Resource Allocation." In *Proceedings of the 2012 Winter Simulation Conference*. C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.: 2690-2704. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

MathWorks. 2016a. *DSP System Toolbox™ Users Guide*. MathWorks®, Release R2016a, Natick, MA.

MathWorks. 2016b. *MATLAB® User's Guilde*. MathWorks®, Release R2016a, Natick, MA.

MathWorks. 2016c. *SimEvents®, User's Guide*. MathWorks®, Release R2016a, Natick, MA.

MathWorks. 2016d. *Simulink® User's Guide*. MathWorks®, Release R2016a, Natick, MA.

MathWorks. 2016e. *Stateflow® User's Guide*. MathWorks®, Release R2016a, Natick, MA.

MathWorks. 2016f. *Writing S-Functions*. MathWorks®, Release R2016a, Natick, MA.

Mosterman, P. J., and J. Zander. 2016. "Cyber-Physical Systems Challenges: a Needs Analysis for Collaborating Embedded Software Systems." In *Software & Systems Modeling* 15(1): 5-16.

Mosterman, P. J., D. E. Sanabria, E. Bilgin, K. Zhang, and J. Zander. 2014. "A Heterogeneous Fleet of Vehicles for Automated Humanitarian Missions." In *Computing in Science & Engineering* 12(3): 90-95.

Schwatinski, T., T. Pawletta, S. Pawletta, and C. Kaiser. 2010. "Simulation-Based Development and Operation of Controls on the Basis of the DEVS Formalism". In *Proceedings of the 7th EUROSIM 2010 Congress*, Prag, Czech Republic.

Seo, K. M., C. Choi, T. G. Kim, and J. H. Kim. 2014. "DEVS-Based Combat Modeling for Engagement-Level Simulation." *Simulation* 2014: 0037549714532960.

Sztipanovits J., Ying S., Cohen I., Corman D., Davis J., Khurana H., Mosterman P.J., Prasad V., and Stormo L. 2013. "Foundations for Innovation: Strategic Opportunities for the 21st Century Cyber-Physical Systems – Connecting Computer and Information Systems with the Physical World." *Report by the Steering Committee for Foundations in Innovation for Cyber-Physical Systems*, National Institute of Standards and Technology (NIST).

Wainer, G. A. 2009. *Discrete-Event Modeling and Simulation.* CRC Press, Taylor & Francis Group, Boca Raton, FL.

Wymore, A. W. 1993. *Model-Based Systems Engineering*. CRC Press.

Zeigler, B. 1984. *Multifacetted Modeling and Discrete Event Simulation.* Academic Press Professional, Inc.

AUTHOR BIOGRAPHIES

WEI LI received his B.S. (1998) and M.S. (2001) degrees in Control Engineering from Tsinghua University, Beijing, China, and his Ph.D. degree (2005) in Systems Engineering from Boston University. From 2006 he has been working in software development at MathWorks, Inc. His current interests include discrete-event and hybrid simulation technologies, quantitative analysis, formal verification and optimization of discrete-event systems, embedded system architecture modeling, network simulation, and operations research. His email address is wei.li@mathworks.com.

RAMMURTHY MANI works in software development at MathWorks, Inc. His interests include simulation technologies, distributed systems and simulation, discrete-event simulation, signal processing, embedded architectures, systems engineering, and computational biology. He received his PhD (1998) in Electrical Engineering from Boston University. His email address is ramamurthy.mani@mathworks.com.

PIETER J. MOSTERMAN is a Senior Research Scientist at MathWorks in Natick, Massachusetts. He also holds an adjunct professor position at the School of Computer Science of McGill University. Prior to this, he was a research associate at the German Aerospace Center (DLR) in Oberpfaffenhofen. He earned his Ph.D. in Electrical and Computer Engineering from Vanderbilt University in Nashville, Tennessee, and his M.Sc. in Electrical Engineering from the University of Twente, the Netherlands. His primary research interests are in Computer Automated Multiparadigm Modeling (CAMPaM) with principal applications in design automation, training systems, and fault detection, isolation, and reconfiguration. In 2009, he received the Distinguished Service Award of *The Society for Modeling and Simulation International* (SCS) for his services as editor in chief of *SIMULATION: Transactions of SCS*. Dr. Mosterman also has been guest editor for special issues on CAMPaM of *SIMULATION*, *IEEE Transactions on Control Systems Technology*, and *ACM Transactions on Modeling and Computer Simulation*. His email address is pieter.mosterman@mathworks.com.