Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

TEACHING DISCRETE EVENT SIMULATION SOFTWARE DESIGN IN THE CONTEXT OF COMPUTER ENGINEERING

James F. Leathrum, Jr.

Dept. of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529, USA

ABSTRACT

Recent events resulted in the consolidation of a degree program in Modeling & Simulation Engineering with a degree in Computer Engineering, though with a major in Modeling & Simulation Engineering. The resulting major strongly highlights the computational aspects of M&S. However, the needs of discrete event simulation in computer engineering have somewhat of a different focus. For instance, the management of simultaneous events is crucial in digital circuit simulation. This paper looks at refocusing a course on discrete event simulation software design to meet the needs of a computer engineering degree while maintaining applicability to the more general community. It discusses modifications in the treatment of models and then mapping those models to software.

1 INTRODUCTION

Old Dominion University has supported Modeling & Simulation Engineering (M&SE) education at the graduate level since the late 90's and at the undergraduate level since 2010 becoming the first ABET accredited M&SE program. Unfortunately, a decision was made to cancel the M&SE undergraduate degree in favor of making a M&SE major under a Computer Engineering degree (Leathrum et al. 2021) to expand the computer engineering program with a computationally focused major. This forced a lot of hard decisions at the curricular level (Leathrum et al. 2022) resulting in a reduction of the number of core M&SE courses while emphasizing the computational concepts.

Two courses to survive the reorganization is Simulation Software Design which teaches data structures, algorithms, software architecture, and programming paradigms in support of simulation executive and application development for discrete event simulation (DES). The course has an obvious prerequisite course focusing on discrete event simulation. However, both courses required a redesign to fit within the context of computer engineering. Computer engineering students are introduced to different models and simulation tools and languages for digital circuit design. The new M&SE courses retain the goal of teaching general DES concepts and applications but required a consideration of the needs of digital circuit modeling and simulation. The resulting courses are novel in computer engineering programs. Hopefully they

- strengthen DES within computer engineering,
- provide a better computational background in DES than found in classical M&S programs based in disciplines such as industrial engineering, and
- give a better DES foundation than found in classical computer engineering programs.

While the goal is a balance, it is true that it is difficult to serve both masters.

This paper first considers the needs of digital circuit modeling and simulation and how DES education can be adapted to include those needs. The needs of digital circuit simulation are balanced with the needs of teaching software design. Simulation worldviews and simultaneous events are the identified as crucial concepts to introduce and illustrate their importance for digital circuit simulation. Then the required

software design concepts to support those concepts are discussed as core topics in the simulation software design course. These courses are scheduled to be first fielded in academic year 2023-2024.

2 BACKGROUND

Old Dominion University created a Modeling and Simulation Engineering undergraduate degree in 2010 (Mielke et al. 2011; Leathrum et al. 2011) after a decade of graduate M&S programs. The degree resulted in the first ABET accredited program of its kind (Mielke et al. 2014; McKenzie et al. 2015; McKenzie 2015). However, changes in academia, in particular a pending drop in enrollments (Kline 2019) and tightening funding, even prior to COVID-19, make it difficult to support niche programs like M&SE going forward. Thus, the decision was made to consolidate M&SE with a larger, more established program. Engineering management and computer engineering were considered within the college of engineering – the decision logic can be found in (Leathrum et al. 2021) – with computer engineering being the final selection. This took advantage of the strong emphasis on computation within the existing degree. But the resulting M&SE major under a degree in computer engineering. But it came with a cost of reducing the core M&S content to account for the required computer engineering content in order to maintain ABET accreditation, but now in computer engineering. (Leathrum et al. 2022) discusses the transitioning of the courses from those under the M&SE degree to the major in computer engineering. This paper goes down another level looking deeper at the computational concepts highlighted in the Simulation Software Design course.

The original M&SE degree had a high computational focus. Leathrum et al. (2019) considers the software skills required by M&S graduates. They highlighted concepts such as object-oriented programming, data structures, open-source software, computer graphics and software architecture. They tied those concepts to application development, simulation tool development, and simulation worldviews.

Leathrum et al. (2017) proposed a discrete event simulation content roadmap starting at the various model paradigms and mapping down to software implementation under the different simulation worldviews. In that process, it was found that event graphs (Schruben 1983) were easy to teach students how to go from the model to a software implementation. Even better, the event graphs gave a convenient visual of the execution of the events from a software point of view for the event scheduling worldview (Collins et al. 2017). Later an equivalent visual was developed for the process interaction worldview (Tracey et al. 2018).

3 DES MODELING & SIMULATION IN COMPUTER ENGINEERING

This section describes M&S in the context of computer engineering. The goal is to highlight the similarities and differences with classical M&S education. The discussion advances top-down, beginning with the high-level modeling process followed in computer engineering and how verification is integrated. Modeling is then discussed as related to simulation worldviews showing how computer engineering education advances from an event scheduling world view to a process interaction worldview. Finally, the importance of simultaneous events is explained in computer engineering, a topic often ignored in classical simulation education.

3.1 Model Refinement and Verification

Modeling in computer engineering tends to follow the hardware design top-down approach. Figure 1a illustrates the Gajski-Kuhn Y-Chart. Design moves around the chart starting with a behavioral description and moving to a structural and then physical description. The design is refined by stepping in a level of the concentric circles when transitioning from a physical description to a refined behavioral description. This synthesis process is illustrated in Figure 1b. Verification is performed by going counterclockwise on a single level or stepping out radially along the same dimension.

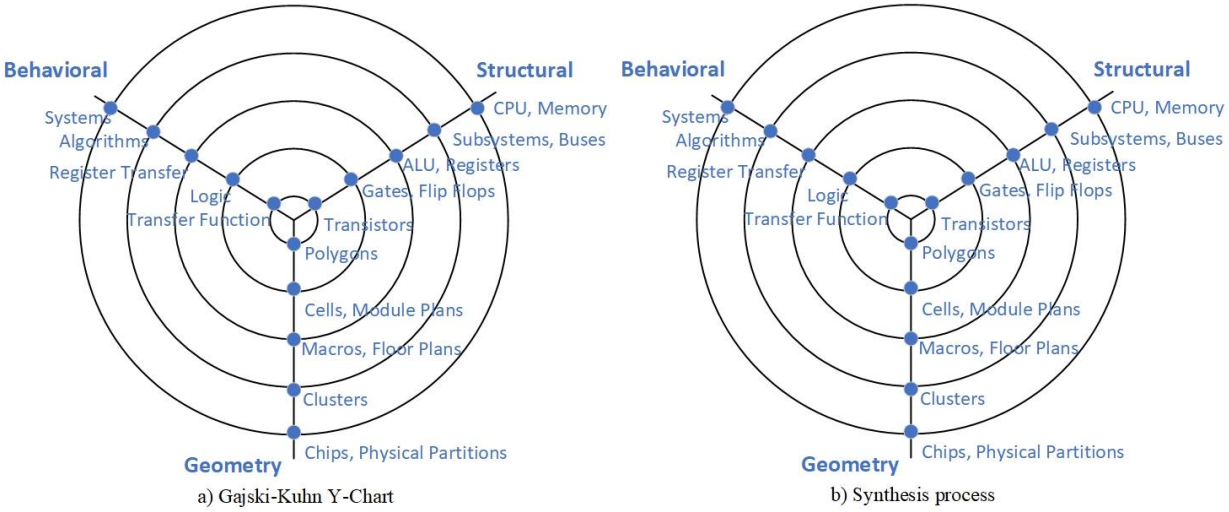

Figure 1. Gajski-Kuhn Y-Chart. Illustrates the domains of modeling refining a model from a high-level abstraction down to a detailed design.

Having students already aware, and hopefully somewhat comfortable, with this structured approach to refinement and verification creates more structure to the verification process than classical simulation education supports. It is believed that students will grasp this concept more easily in the computer engineering domain and then be able to apply it better to general applications.

3.2 World Views

Computer engineering simulation education comfortably resides in two world views, event scheduling (or event based) and process interaction. Event scheduling is appropriate based on early computer engineering courses and process interaction for advanced course. Each must be introduced in the curriculum to support understanding of the different levels of modeling. Classically M&S is taught in terms of event scheduling, however, Rashidi (2017) surveyed simulations software and found that approximately 50% supported process interaction, 40% event scheduling, 20% activity scanning, and 10% three-phase (recognize some software supports multiple world views). This indicates a requirement for properly covering process interaction.

3.2.1 Event Scheduling

Early in computer engineering education, students are introduced to modeling through state machines and gate-level circuit design. They are taught that state variables are discrete, simply Boolean early in learning, that change at discrete moments in time. In a clocked circuit, students are introduced to a behavioral model where all state variables changed simultaneously based on a clock edge, generally the rising edge. State machines are taught similarly, that at discrete moments in time, system state transitions to a new state based on inputs and current state. This approach is easily introduced in simulation as event scheduling.

The author's experience shows that event graphs (Schruben 1983; Buss 1996) are an excellent representation of event scheduling, easily understood by students as they succinctly represent both the scheduling of events and the modification of state variables (Leathrum et al. 2017). It will be later discussed that it is a convenient model for a software model to teach software design of DES.

3.2.2 Process Interaction

In advanced digital circuit courses, students are introduced to High-Level Description Languages (HDL). In this discussion the Very High-Speed Integrated Hardware Description Language (VHDL) (IEEE 2019) will be utilized but a similar discussion could be held for Verilog (IEEE 2005).

VHDL provides a software programming approach to hardware development. The hardware can be described at the various levels of the Y-chart, starting with a behavioral level and refining to a structural description. Simulation is supported along the way and synthesis supports generating hardware directly from the code.

As code, VHDL does not readily map to the event scheduling worldview, but rather is better supported by process interaction. The language supports the passage of time through three basic statements: wait on, wait until, and wait for. Wait on passes time until a designated signal changes value. Wait until passes time until a condition is met. Wait for passes time for a specified amount of time. These are similar in nature to simulation languages, for example the wait statement in SystemC (IEEE Computer Society 2012). Figure 2 demonstrates VHDL code's similarity to process interaction simulation code where the "wait for 10 ns" suspends execution of the process P1 for 10 nanoseconds.

```
P1: process
begin
   signal value : std_logic_vector(7);
   signal nextvalue : std_logic_vector(7);
   data <= conv_std_logic_vector(1, width);
      loop1: for i in 1 to 255 loop
             value <= std_logic_vector( unsigned(value) + 1 );
             nextvalue <= std_logic_vector( unsigned(value) + 2 );
          wait for 10 ns;
   end loop;
end process;
```

Figure 2. Example VHDL code illustrating its similarity to process interaction simulation code.

3.3 Simultaneous Events

Classical M&S commonly ignores the concept of simultaneous events, with the possible exception of distributed simulation (Cota and Sargent 1990; Peschlow and Martini 2007; Kim et al. 1997). While it is understood that the modeling of simultaneous events can have an impact on simulation results (Peschlow and Martini 2010; Carboni et al. 2019; Dinila 2021; Raczynski 2021), the discussion is largely avoided in education. However, simultaneous events are a common concept in digital circuit simulation as activity is initiated by a single clock signal. Especially when doing behavioral modeling, simultaneous events can dominate the simulation. The simple example in Figure 3 illustrates the concept. The circuit is comprised of two subcircuits, each controlled by a clock. The clock dictates when the output of each subcircuit changes, thus the outputs will change simultaneously, especially when behaviorally modeling. If one subcircuit changes its output first and then the second circuit evaluates its output, the second circuit could be using an incorrect input. Thus, students are introduced to the concept of a current circuit state and a next circuit state. The next circuit state is solely defined by the current circuit state to insure a consistent view of the inputs.

Note that the standard M&S solution to simultaneous events is for the model to provided tie-breakers so the events logically still execute sequentially or to accept a predefined ordering, usually the order the events were scheduled. But the tie-breaker approach requires extensive model modification to support the concept of current state/next state in digital circuits. It is best to define this behavior as the expected

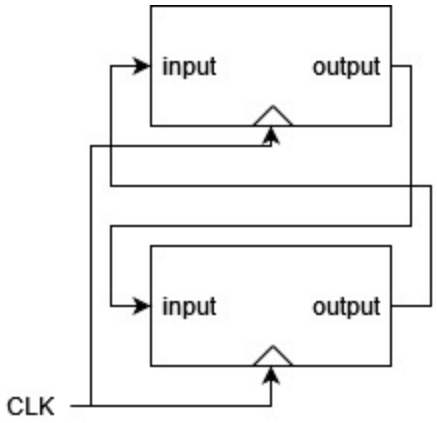

Figure 3. Simple sequential circuit. The CLK signal results in simultaneous events at each logic gate.

behavior of simultaneous events and include it as a core concept in the simulation executive. And while a predefined ordering is easy, it generally will result in incorrect results for digital circuits.

The impact of simultaneous events is highlighted when considering HDL programming. VHDL defines two modes of defining information, variables and signals. They differ in where they can be used in the code, but more importantly, they differ in when value changes occur during execution. A variable acts like a normal programming variable in that when assigning to the variable, the variable takes on the new value immediately. However, signals do not change immediately, rather wait until the next event associated with its process. In the example in Figure 2, the values of val and nextval do not change when the assignment is executed, rather they change on the wait statement. If the integer equivalent of val was 5, then the val assignment would record that val was to be modified to 6 at the next time stamp. The same would be true for the nextval assignment, except it would use the value of 5 for val since val has not been updated yet. If val' is defined as the next state of val and nextval' is defined as the next state of nextval, then the assignments can be considered as:

```
val' <= f1(val);
nextval' <= f2(val);
```

and the update on the wait statement would be

```
val <= val';
nextVal <= nextval'
```

This gets more confusing if there are multiple assignments to the same signal. The last assignment would have effect and all previous would be ignored. Thus, the logic that modifies value in Figure 4 would have value increased by 3, not 4, since only the second assignment would have an effect.

4 DES SOFTWARE DEVELOPMENT IN COMPUTER ENGINEERING

4.1 Prior DES Software Development Education

A second course in the curriculum teaches software development of discrete event simulations. In the past the course focused on classical DES with an emphasis on queuing models. Since classical DES education is based on event scheduling, the majority of application development utilizes event graphs as a convenient point from which to develop an equivalent object-oriented software model leading to a C++ implementation. The application/simulation executive architecture was taught (Pidd 2004) to then allow a

```
P1: process
begin
    signal value : std_logic_vector(7);
    data <= conv_std_logic_vector(1, width);
    loop1: for i in 1 to 255 loop
        value <= std_logic_vector( unsigned(value) + 1 );
        value <= std_logic_vector( unsigned(value) + 3 );
        wait for 10 ns;
    end loop;
end process;
```

Figure 4. VHDL code where a single signal has values assigned to it twice and only the second assignment has an effect.

discussion of design of the simulation executive. A behavioral model was developed followed by interface design, algorithmic development, and several approaches to the future event list (linked list, skip list, calendar queue, lazy queue, ladder queue, heap, and binary search tree). Students developed their own simulation executive, several applications of queuing models, and a library of tasks (source/sink, delay, condition, resource request/release, fork/join, etc.) similar to those available in most visual-based commercial simulation tools.

The education was rounded out by a discussion of implementing a process interaction worldview in software, a programming paradigm not comfortable to the students as it violates the basic programming concepts they have been previously taught. But they were not taught the development of an associated simulation executive as they did not have the skillset to program a process-based program.

4.2 Concept Changes Moving to a Computer Engineering Based Curriculum

Moving to computer engineering resulted in two primary modifications to the introduction of software development: process interaction worldview simulation executive development and simultaneous events. The importance of both concepts was previously discussed in Section 3. The basic structure of the rest of the content is unchanged as the event scheduling worldview is still relevant to simulation of state machines and gate-based digital circuits. But to fit the new content, other content was reduced, not eliminated. For instance, the variety of future event list implementations considered was greatly reduced to only a linked list (retained for its simplicity to allow the student to get an initial simulation executive working quickly), a calendar queue, and a heap, sufficient coverage to discuss the performance tradeoffs.

4.2.1 Process Interaction Worldview

One of the difficulties in teaching the implementation of a process interaction worldview simulation executive in the past was the lack of support for coroutines in programming languages and the impracticality of a thread-based implementation (the overhead overwhelms the performance despite only one thread being active at a time). Coroutines were introduced in the 1960's (Conway 1963) but with little general language support until recently. Current languages supporting coroutines include Python, C++, Go and Rust with the introduction to C++ being very recent.

Coroutines are a known approach to implementing process interaction simulations. Simula is considered the first language to utilize coroutines in support of simulation (Dahl and Nygaard 1966). More recent approaches include: 1) utilizing a language with direct coroutine support (Rust) (Weber and Fischer 2020), 2) library supported (Xu and Li 2012), and 3) modifying a Java VM (Weatherly and Page 2004).

Educationally it is difficult to include coroutines in the discussion of simulation software as computer science curricula rarely teach it thus requiring it to be taught prior to introducing its use in simulation. However, their importance in supporting digital circuit simulation increases the necessity to spend the time

introducing the concept. It is hoped that computer science curricula will consider teaching coroutines in the future relieving the pressure of the increased content when teaching simulation software.

The selection of a coroutine language to introduce coroutines was problematic. Python was considered an inappropriate language for performance reasons. Java requires library support or modification to the VM (Weatherly 2004); introducing a library is an extra learning overhead. Go and Rust were not considered as not being offered within either the computer science or computer engineering curricula at Old Dominion University. But now that coroutines are part of the C++ language (Mazières 2021), an attempt is in progress to introduce coroutines in the course sufficiently to allow implementation of the simulation executive.

Coroutines are beneficial for teaching process interaction programming as they support the suspension of a function until reactivated. Unlike threads, only one coroutine can be active at a time, thus they do not support concurrency, rather are a different programming paradigm. However, once a coroutine is executed, it maintains its state of execution until terminated (stackless coroutines) (Weber and Fischer 2021). Therefore, a coroutine does not incur the overhead of function calls. This results in faster execution for many properly structured programs. Applying the concept to VHDL-like circuit simulation, each process in VHDL can be implemented by a coroutine. Once a coroutine is suspended, a reference to that coroutine is scheduled as part of an event for activation of the coroutine at the appropriate simulation time or condition.

Students will be taught the basics of coroutines in C++, not a thorough coverage. They will learn how to initiate a coroutine, get a reference to a coroutine, and suspend/activate a coroutine. They will then be shown how the only modification to the simulation executive is how an event is initiated, either by a function call for event scheduling or by a coroutine activation for process interaction. They will then be shown the ease with which they can now combine the two creating a simulation. For instance, part of a circuit, possibly a state machine, can be implemented using event scheduling while another part can use process interaction in the same simulation execution.

4.2.2 Simultaneous Events

The simultaneous event model utilized in digital circuit simulation can be addressed at the model level. However, with its prominence in circuit simulation it is more appropriate to provide direct support. A straightforward approach to the problem similar to the delayed commit in (Mathew 2007) will be introduced to the students. In this approach, polymorphism is utilized so all state variables inherit a class DelayedCommit much as VHDL introduces the type *signal*. When a modification is made to a state variable, the DelayedCommit can register itself with the simulation executive. Only the first modification results in being registered, but any successive modifications result in a change to the state variable to mirror the behavior of a *signal* in VHDL where only the last modification is accepted. Then when the simulation executive recognizes that it is about to advance simulation time, it commits all state variable modifications by calling a SetCommit virtual member function of DelayedCommit to actually update the state variable value. This concept is similar to the three-phase worldview allowing the introduction of that worldview time permitting, though this is not anticipated due to the extra requirements put on the course (Leathrum et al. 2022).

5 FUTURE WORK

Courses in discrete event simulation and discrete event simulation software design are being first offered in the 2023-2024 academic year. The courses will have the first cohort of computer engineers in the M&S major. As such, an assessment will be performed, as required for all core courses, identifying weaknesses in the material. After reviewing these results, the material will be modified to strengthen prerequisite knowledge and to modify how the material is introduced to improve student success.

The biggest challenge in supporting the coursework is the lack of an academic base. For instance, there are no textbooks covering this material, either from M&S, computer engineering, or computer science. Clearly the base knowledge exists as the concepts are in practice, but the lack of academic support puts a

burden on the instructors to create that material. And when portions of the material, for example coroutines, are lacking in their explanations in the literature. And while coroutines are entering the mainstream, they are still immature. In C++, community feedback suggests a series of revisions are necessary to make them truly useful.

In addition, a serious recruiting effort is underway. Recruiting was always a challenge with the previous degree program as potential students did not understand M&S. The hope is that computer engineering will initially attract the students and then the advanced computational flavor of the new major will attract students. However, there is concern that the percentage of female students will drop relative to the previous degree (approximately double the engineering average) as computer engineering has never been that high.

6 CONCLUSIONS

While computer engineering has always relied on DES, little crossover with the classical DES discipline exists. The merging of a more classical M&SE degree with a computer engineering degree has forced a evaluation of the differences and similarities in order to strive to serve both communities at an undergraduate level. This paper has highlighted the similarities and differences. While the original degree was very computationally intensive, a shift was required to align with the needs of computer engineering. However, it is believed that this shift just emphasizes concepts not normally covered in M&S courses, for example the required focus on the management simultaneous events, and not leaving their management to the model developer. The result is an intensive software experience exposing students to advanced concepts such as coroutines. However, it is the author's opinion that coroutines demonstrate a general utility, especially their performance relative to classical function-based programming, which will result in them becoming more mainstream in computer science curricula, thus easing the burden on the course in future years. The course will first be taught in Spring 2024 with the discrete event simulation prerequisite course first being taught in Fall 2023. All supporting software support for the course has been developed and tested ready for deployment.

Future work will need to consider the impact of this work on an existing course in distributed simulation. While there is little publication on DES in computer engineering, there is a fair amount considering parallel DES of digital circuits (Meraji and Tropper 2012; Patrou et al. 2019) and even applied to education (Rossainz-Lopez 2019). And again, the importance of concepts like simultaneous events need to be focused on in the parallel context.

REFERENCES

Buss. A. 1996. "Modeling with Event Graphs". In *Proceedings of the 1996 Winter Simulation Conference*, edited by J. Charnes, D. Morrice, D. Brunner, and J. Swain, 153-160. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Carboni, J., D. Hodson, J. Miller, and J. Millar. 2019. "The Effect of Modeling Simultaneous Events on Simulation Results". In *Proceedings of the International Conference on Scientific Computing (CSC)*, edited by H. Arabnia, L. Deligiannidis, M. Grimaila, D. Hodson, and F. Tinetti, 112-122. Athens, Greece: CSREA Press.

Collins, S., N. Gonda, L. Dumaliang, J. Leathrum, and R. Mielke. 2017. "Visualization of Event Execution in a Discrete Event System". In *50th Annual Simulation Symposium (ANSS 2017)*, edited by S. Jafer and J. Padilla. Article 7, 1-12. San Diego, CA: Society for Computer Simulation International.

Conway, M. 1963. "Design of a Separable Transition-Diagram Compiler". *Communications of the ACM* 6(7):396-408.

Cota, B. and R. Sargent. 1990. "Simultaneous Events and Distributed Simulation". In *Proceedings of the 1990 Winter Simulation Conference*, edited by O. Balci, R. Sadowski, and R. Nance, 436-440. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Dahl, O., and K. Nygaard. 1966. "SIMULA: An ALGOL-Based Simulation Language". *Communications of the ACM*, 9(9):671-678.

Dilinila, B. 2021. *A Behavioral Model for Simultaneous Event Execution in Sequential Discrete Event System Simulations*. Masters thesis, Department of Computational Modeling and Simulation Engineering, Old Dominion University, Norfolk, VA, https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1060&context=msve_etds, accessed 5th April 2023.

Grout, I. 2008. *Digital Systems Design with FPGAs and CPLDs*. Newton, MA, USA: Newnes.

IEEE Computer Society. 2012. *IEEE Standard for Standard SystemC Language Reference Manual*. IEEE Std 1666-2011.
IEEE. 2005. *IEEE Standard for Verilog Hardware Description Language*. IEEE Std 1364-2005.
IEEE. 2019. *IEEE Standard for VHDL Language Reference Manual*. IEEE Std 1076-2019.
Kim, K., Y. Seong, T. Kim, and K. Park. 1997. "Ordering of Simultaneous Events in Distributed DEVS Simulation". *Simulation Practice and Theory* 5(3):253-268.
Leathrum, J., R. Mielke, A. Collins, and M. Audette. 2017. "Proposed Unified Discrete Event Simulation Roadmap for M&S Curricula". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer and E. Page, Article 357, 1-12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Leathrum, J., Y. Shen, M. Sosonkina, and M. Audette. 2022. "Transitioning M&S Courses from a M&S Engineering Degree to a Major Under a Computer Engineering Degree". In *Proceedings of MODSIM World 2022*, May 9th-11th, Norfolk, VA. Paper 19, 1-11.
Leathrum, J., Y. Shen, O. Gonzales. 2021. "A New M&S Engineering Program with a Base in Computer Engineering. In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Massoud, Z. Zheng, C. Szabo, and M.Loper, 3285-3294. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Leathrum, J., J. Sokolowski, Y. Shen, and M. Audette. 2019. "Software Skills Required by M&S Graduates for DES Development."
Mathew, R. 2007. *The Distributed Independent-Platform Event-Driven Simulation Engine Library (DIESEL).* Ph.D. dissertation, Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, Virginia, https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1100&context=ece_etds, accessed 20th April 2023.
Mazières, D., 2021. "My Tutorial and Take on C++ 20 Coroutines". https://www.scs.stanford.edu/~dm/blog/c++-coroutines.pdf. Accessed 6th April 2023).
Meraji, S. and C. Tropper. 2012. "Optimizing Techniques for Parallel Digital Logic Simulation". *IEEE Transactions on Parallel and Distributed Systems* 23(6):1135-1146.
Patrou, M., J.-P. Legualt, A. Graham, and K. Kent. 2019. "Improving Digital Circuit Simulation with Batch-Parallel Logic Evalution". In *2019 22nd Euromicro Conference on Digital System Design (DSD)*. August 28th-30th, Kallithea, Greece, 144-151.
Peschlow, P., and P. Martini. 2007. "Efficient Analysis of Simultaneous Events in Distributed Simulation". In *11th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)*. October 22nd-26th, Chania, Greece, 244-251.
Peschlow, P., and P. Martini. 2007. "A Discrete-Event Simulation Tool for the Analysis of Simultaneous Events". In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2007*, October 22nd-27th, Nantes, France.
Pidd, M. 2004. "Simulation Worldviews – So What?". In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. Ingalls, M. Rossetti, J. Smith, and B. Peters, 288-292. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Raczynski, S. 2021. "Simultaneous Events, Singularity in the Space of Models and Chicken Game". *International Journal of Modeling, Simulation, and Scientific Computing* 12(4).
Rashidi, H. 2017. "Discrete Simulation Software: A Survey on Taxonomies". *Journal of Simulation* 11(2):174-184.
Rossainz-Lopez, M., C. Ceron-Ganica, E. Archundia-Sierra, P. Cervantes-Marquez, D. Carrasco-Limon, and B. Sanchez-Rinza. 2019. "Parallel Simulation of Digital Logic Circuits Using Message Passing via CSP as an Educational Tool". In 5th *Iberoamerican Workshop on Human-Computer Interaction.* June 19th-21st, Puebla, Mexico, 284-298.
Schruben, L. 1983. "Simulation Modeling with Event Graphs". *Communications of the ACM* 26(11):957-963.
Tracey, T., B. Dilinila, J. Leathrum, and R. Mielke. 2018. "Visualization of the Process Interaction Worldview in Discrete Event Simulation". In *Proceedings of MODSIM World 2018*, May 22nd-23rd, Norfolk, VA, Paper 59, pp 1-12.
Weatherly, R., and E. Page. 2004. "Efficient Process Interaction Simulation in Java: Implementing Co-routines within a Single Java Thread". In *Proceedings of the 2004 Winter Simulation Conference*, edited by R. Ingalls, M. Rossetti, J. Smith, and B. Peters. 1437–1443. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Weber, D., and J. Fischer. 2020. "Process-Based Simulation with Stackless Coroutines". In *Proceedings of the 12th System Analysis and Modeling Conference (SAM '20)*, October 19th-20th, 84-93.
Xu, X., and G. Li. 2012. "Research on Coroutine-Based Process Interaction Simulation Mechanism in C++". In *Proceedings of AsiaSim 2012*, edited by T. Xiao, L. Zhang and M. Fei, Communications in Computer Information Science. 325:178-187.

AUTHOR BIOGRAPHIES

JAMES F. LEATHRUM, JR. is an Associate Professor in the Department of Electrical and Computer Engineering at Old Dominion University. He earned the Ph.D. in Electrical Engineering from Duke University. His research interests include simulation software design, distributed simulation, simulation-based test & evaluation, and simulation education. His email address is jleathru@odu.edu.