HYBRID AGENT-BASED AND DISCRETE EVENT SIMULATION IN MASON

Giuseppe D'Ambrosio
Department of Computer Science
Università degli Studi di Salerno
Via Giovanni Paolo II, 132
Fisciano, SA, Italy
gdambrosio@unisa.it

Sean Luke
Department of Computer Science
George Mason University
4400 University Drive
Fairfax, VA, USA
sean@cs.gmu.edu

ABSTRACT

Agent-Based Modeling and Discrete Event Simulation are two of the most common simulation approaches, and are supported by many simulation tools. Recently, modelers have begun integrating these methods to simulate real-world situations which have some elements better represented by DES and others better represented by ABM. Hybrid ABM and DES simulation is a growing trend but there is a lack of simulation software supporting it, and effectively no open-source software at all. This paper introduces a new open-source DES extension to MASON, a widely used ABM tool, to implement both DES and hybrid ABM-DES models. The goal of the system is to help researchers realize hybrid simulations while exploiting the same characteristics that led to MASON's success, notably its efficiency, extensibility, and ease of integration with other tools. The MASON DES extension is still in its infancy but is already being employed in a large simulation project studying malicious attacks affecting supply chains.

Keywords: Hybrid Simulation, Discrete Event Simulation, Agent-Based Model, Simulation Framework

1 INTRODUCTION

Over the last three decades, dramatic increases in computing power have transformed simulation options in fields such as the social sciences and population biology. These fields study the complex and stochastic interactions of potentially very large numbers of actors with sophisticated behaviors, and computing brawn has only recently enabled researchers to move beyond abstract systems dynamics models to simulations directly modeling the individual actors themselves. Two of the most common simulation approaches have been *Discrete Event Simulation* (DES) and *Agent-based Modeling* (ABM). These methods are normally applicable to quite different problem domains, and there are many available simulation tools specifically for one or the other modeling technique. However, relatively recently we have seen *Hybrid Simulation* (HS) models which seek to combine both methods in a single simulation.

DES is a modeling approach where the system's state changes at discrete points in time called *events*, updating the system only when necessary. One very common DES approach is to model a system as a state graph between actors called *processes*, with events being regular discrete interactions among the processes. DES systems are often used to model supply chains, transportation, and communications networks (Franceschini et al. 2014). Notably, in a DES model, if nothing is changed between events, no computation is required, making it a computationally efficient approach (Zhang et al. 2011).

[0]**Text version 2.** This version has slight corrections to the M/M/1 Queue example compared to the original publication.

ABM is similar to DES in many ways: it also involves discrete events and relationships among actors, here called *agents*. However, rather than relying on a fixed or limited set of interactions, an ABM is used to model arbitrary interactions among a very large number of agents, each following simple interaction rules (Antelmi et al. 2022, Macal and North 2010). For example, a bird in a flock may rely on a simple set of behaviors to interact with its neighbors while flying, but *which* neighbors it interacts with may change rapidly over time as the flock structure dynamically reconfigures. ABMs are commonly used to model swarms and very large groups, social networks, and other scenarios where a small set of simple rules can give rise to complex and unpredictable macrophenomena (Chan et al. 2010).

Many real-world situations have characteristics that do not fit into a single simulation approach, but they have different aspects best represented by different methods (Heath et al. 2011). This need has given rise to a growing trend named Hybrid Simulation (Mittal and Krejci 2017) that attempts to merge multiple simulation methodologies into one model (Eldabi 2021). HS allows modelers to obtain higher flexibility, to work at different abstraction levels, and to exploit the advantages of different approaches (Wang et al. 2014). One increasingly common form of HS integrates DES and ABM, allowing researchers to easily realize models involving agents' autonomous behavior and decision processes. For this reason, the hybridization of ABM and DES was listed among the simulation technology priorities in the survey of Taylor and Robinson (2006).

Despite its rising popularity, there is a lack of software, particularly open-source software, for hybrid ABM-DES simulations. Modelers who want to adopt this approach need to develop their solution from scratch or rely on AnyLogic (AnyLogic 2000), the only commercial software which provides both paradigms. This scarcity is an important gap because developing an HS model often requires more time and skills than applying each method separately (Eldabi 2021).

In this paper, we introduce an extension to MASON, an Agent-Based Modeling toolkit, which incorporates Discrete Event Simulation into ABM models. The objective is to help researchers seeking to integrate both methods, or aspects of them, in the same simulation. Our MASON DES facility is still in its infancy, and we expect it to be further fleshed out over the coming years, but it has already served as the foundation for a large funded research study examining how criminal networks may attack critical supply chains.

The remainder of the paper is organized as follows. Section 2 summarizes the literature regarding DES and ABM and provides an overview of HS. Section 3 gives background on MASON as a simulation toolkit. Section 4 introduces the MASON DES extension, describing its main components and functionality. Section 5 illustrates how to use the MASON DES extension, first showing a simple M/M/1 queue example, then discussing a model we have developed for our supply chain project. Section 6 compares MASON against AnyLogic, the most adopted ABM-DES toolkit in the literature. Section 7 concludes with future directions.

2 BACKGROUND AND PREVIOUS WORK

2.1 Agent-based Modeling

An Agent-based Model starts with one or more entities called *agents*, through which the modeler can shape the global behavior of a complex system (Antelmi et al. 2022). An agent is an autonomous and independent entity, capable of making decisions based on its surrounding environment and interactions with other agents. The modeler shapes the behavior of agents with specific rules and defines the environment delineating how they can move and interact. ABM enables researchers to understand the overall behavior of a system by defining its micro-level properties, i.e., the behavioral rules of its agents, and observing emergent macrophenomena (Macal and North 2010).

The seminal ABM tool was SWARM (Minar et al. 1996), released in 1997. SWARM places agents on a schedule to be woken up at various times and allowed to do work; it also defines one or more physical en-

vironments, often grids, in which objects and data — and commonly the agents themselves — exist, which agents can manipulate, and which provide a substrate on which the agents can interact with one another. Many later ABM tools have largely been derivative of the SWARM approach. Nowadays, the three most well-known ABM toolkits are NetLogo, Repast, and MASON. MASON is discussed in Section 3. In NetLogo (Wilensky 1999), the modeler uses a graphical interface to place, define, and program individual agents. NetLogo is widely used among beginners and less experienced coders. Repast (Nick Collier 2001) is an open-source family of ABM simulation platforms, the most well-known being Repast Simphony (North et al. 2013). Repast Simphony is written in Java and structured somewhat like MASON, though it tends to couple visualization much more tightly to the model than MASON does.

2.2 Discrete Event Simulation

Discrete Event Simulation attempts to model (unsurprisingly) discrete events during which processes interact with one another, often along edges in a graph. DES tends to be fixed in structure and thus more easily formalized and analyzed than ABM. A wide range of commercial and open-source DES tools exist.

Commercial DES software tools are often easy to use and provide graphical modeling functionality, but are expensive, limited for complex and non-standard applications, and offer poor support for advanced developers (Hlupic 1999). Open-source DES tools are an alternative but often lack many capabilities and features, leading most modelers to adopt commercial software (Vieira et al. 2019). Major commercial DES tools include *SIMUL8* (SIMUL8 2003), *Arena* (Arena 2000), *OMNeT++* (OMNeT++ 2001), and *AnyLogic* (AnyLogic 2000). For our purposes, AnyLogic is particularly notable because it is a multi-method simulation modeling tool supporting ABM, DES, and System Dynamics simulation, and hybrids of the same.

State-of-the-art of open-source DES tools offer many more options than commercial solutions, but many have significant problems. Some were developed for a specific use case, study, or scientific article and are no longer supported, while others have never been employed in real use cases. Moreover, Dagkakis and Heavey (2016) found that most open-source DES software is released with no documentation and fails to adopt any version control, placing another obstacle to its adoption. Some of the more successful, and properly maintained, open-source DES tools include *DESMO-J* (Page and Kreutzer 2006), *JaamSim* (King and Harrison 2013), *SimPy* (Klaus G. Müller and Tony Vignaux 2002), and *VLE* (Quesnel et al. 2009).

2.3 Hybrid Simulation

Brailsford et al. (2010) argue that trying to represent a complex scenario with only one simulation method, although possible, is like "a case of hammering in a screw" since the modeler is forcing the problem into the simulation method, instead of adapting the method to the problem. Combining different kinds of models to realize a heterogeneous one is a well-known problem; in 2001, Ptolemy II (Liu, Liu, and Lee 2001) pioneered the definition and execution of heterogeneous models adopting a hierarchical system. HS originated from the need to study more complex problems by combining different simulation approaches in a complementary manner. The combination of DES and ABM is an increasingly common HS approach (Huanhuan et al. 2013, Mittal and Krejci 2017, Taylor and Robinson 2006), as the two methods are both popular and can work well together.

DES and ABM share many characteristics, notably the involvement of stochastic elements, but differ in where and how the modeler defines behavioral rules. DES uses system-level rules to govern entities' actions and movements. This allows more formalism and better statistical insights and fits well with well-understood systems. On the other hand, DES complicates shaping intelligent entities with peculiar movements and interactions, making it very difficult to model social behaviors. In contrast, ABM places the agents' logic

directly within them, and so is well-suited to simulate hard-to-predict macro-phenomena arising from the agents' complex micro-level interactions and behaviors.

The advantages of each method are complementary, and so hybrid ABM-DES simulations allow the modeler to define individual behaviors and movements while simulating organizational processes using a process-oriented approach. This combination enables researchers to easily represent human movement, individual decision-making, and social interactions. Many real-world problems have some elements that are easy to represent with a DES and others with an ABM. For instance, process-based activities that include resource management but are involved in social interactions (Heath et al. 2011) (thus benefitting from a hybrid of DES and ABM) are commonly required in domain areas like health care, supply chains, transportation and logistics, and manufacturing (Brailsford et al. 2019, Rondini et al. 2017, Sumari et al. 2013).

Different approaches have been used over the years to realize HS. One simple method is to develop different models separately and then feed the output of one model into the input of the other. This is feasible only in a limited number of scenarios and introduces the problem of correlating and managing the two models. An alternative is to develop some mixture of the two methods in a single one, but the lack of library software supporting it hinders this approach (Langarudi et al. 2021) even if some progress has been made (Heath et al. 2011). Researchers have developed ABM-DES hybrid models by developing tools entirely from scratch (Farsi et al. 2019, Furian et al. 2014, Mittal and Krejci 2017, Zhang et al. 2011), by combining different existing simulation libraries (Huanhuan et al. 2013, Nouman et al. 2013), or, most perhaps often, by using AnyLogic (Cimini et al. 2021, Rondini et al. 2017, Vempiliyath et al. 2021, Viana et al. 2018).

3 MASON

MASON (Luke et al. 2005) is a popular open-source Java agent-based modeling toolkit in active development since 2003. It is designed to be fast, hackable, modifiable, and easily integrated with other tools. MASON has been used very widely over the last twenty years by many modelers and institutions.

MASON carefully delineates between the model and its visualization, and while this is common in the simulation field at large, it is unusual among agent-based modeling toolkits. A MASON model is encapsulated in a singleton object called a *SimState* and knows nothing of visualization or inspection. To visualize, inspect, and graphically manipulate the model, the modeler builds a separate *GUIState* which references the SimState objects and displays their visualization. This allows MASON models to run without any visualization, then have different kinds of visualization attached dynamically. It also permits MASON models to be frozen mid-run, serialized to disk or transferred to another computer or environment, and resumed there. MASON models are entirely self-contained. They can be run in parallel in separate threads or even in the same thread in a process: a MASON model can even contain additional MASON models run recursively inside it. For example, agents might make decisions in a model by running their internal simulations.

MASON models contain two primary components. First, a model holds a representation of time in the form of a heap-based schedule. Agents are scheduled to be woken up at some real-valued time in the future in order to perform some action of their choosing. Agents can then reschedule themselves on the schedule. Second, the model contains one or more representations of space called *fields*. The modeler may provide his own custom fields, but many common ones are available, including network and graph structures, many kinds of grids, continuous space in bounded, unbounded, or toroidal 2D or 3D space, and GIS environments via the *GeoMASON* extension (Keith Sullivan and Mark Coletti and Sean Luke and Andrew Crooks 2010). For each field provided, MASON also provides an accompanying set of 2D and 3D visualization and inspection facilities. MASON has many additional features, including parameter sweeping, publication-quality charts and graphs, and integration with massively distributed model optimization tools. Finally, for larger models, *Distributed MASON* (Cordasco et al. 2018) and *Distributed GeoMASON* can be used to distribute a model over a large number of processors in a cloud computing or server farm environment.

4 THE MASON DES EXTENSION

The MASON DES extension aspires to provide an alternative to the existing hybrid ABM-DES simulation software by implementing DES facilities on top of MASON's ABM engine and fully integrating them with it. The DES extension is intended to be efficient, easily modifiable, and extensible, offering the same characteristics as MASON itself. The MASON DES extension has been designed to support large-scale simulations and is currently used in a major research and industrial use case.

The DES system essentially consists of a graph structure of *processes*, any of which may, in response to some *event*, communicate with one or more other processes primarily by offering *resources* to them or requesting the same. Resources are used both to transfer information or ownership from one process to another and also serve as the elements seized and released from common resource pools to implement semaphores. Resources are typed, and the modeler may define additional types, but they are grouped into three major categories. First, there are *countable* resources, such as money or bricks. Second, there are *uncountable* resources, such as gasoline or happiness. Third, there are basic tokens called *entities*, which can also contain collections of resources and a manifest, and so might additionally be used to model composite things like cargo containers of various resources.

The DES system defines a non-blocking push-style flow graph with optional pull. That is, during an event, a process may offer to transfer one or more resources to a downstream process. The downstream process must immediately accept or refuse some or all of them. A downstream process can also optionally request that an upstream process offer it some range of resources, and the upstream process may refuse this request.

Processes may take several forms. *Sources* only offer resources, *sinks* only receive resources, and *middlemen* both offer and receive resources. Many provided middlemen, called *filters*, are designed to transform or act on a received resource and immediately offer it or some transformed version of it to other processes. Most processes can only receive a single resource type and offer a single type, but so-called *multi* processes are provided to receive and offer many types. For example, a factory might be implemented as a multi-process that accepts steel, electricity, and tires and produces bicycles and waste. Modelers can choose to use provided process classes or to subclass and customize any of them as they prefer.

Every process is implemented as a MASON agent: it can be placed on MASON's schedule to be woken up at a specific time (its event) to do work. However, only certain processes, such as *delays* or resource generators (*sources*), need to be placed on the schedule. Instead, many processes are designed to act only when offered resources. For example, when a *queue* — essentially a warehouse — receives an offer of resources, it stores them in the queue if it has enough capacity. Then it immediately offers resources from the queue to select downstream processes. While a queue can also be scheduled to offer to downstream processes at specific times, like every three hours, by default it only offers them when it has received new resources.

MASON also offers *macros*: process-like objects that encapsulate subgraphs of other processes. For example, the graph defining a university computer network may be encapsulated into a macro, then placed in a higher-level graph of abstracted institutional networks. Macros are fully recursive.

A MASON DES extension model is built exactly like a MASON ABM model: it uses the same SimState and separation between model and visualization. The DES extension offers rudimentary debugging, inspection and probing, and visualization, which we intend to improve over time. Moreover, it inherits MASON's checkpointing and other capabilities, including integration with external tools such as ECJ (Luke 1998). The primary MASON feature not available to the DES system is the ability to be distributed over multiple processors in Distributed MASON. However, a Distributed MASON model can have one or more DES graphs on each processor or contained within agents migrating through the model.

```
1  public class MM1Queue extends SimState {
2      public MM1Queue(long seed) { super(seed); }
3
4      public void start() {
5          super.start();
6
7          Entity entity = new Entity("foo");
8
9          Source source = new Source(this, entity);
10         source.setRateDistribution(new Exponential(15, random));
11         source.autoSchedule(true);
12
13         Pool pool = new Pool(new CountableResource("server", 1), 1);
14         Lock lock = new Lock(this, entity, pool, 1);
15         source.addReceiver(lock);
16
17         Delay delay = new Delay(this, entity);
18         delay.setDelayDistribution(new Exponential(15, random));
19         lock.addReceiver(delay);
20         lock.setSlackProvider(source);
21         lock.setSlackReceiver(lock);
22
23         Unlock unlock = new Unlock(this, entity, pool, 1);
24         delay.addReceiver(unlock);
25
26         Sink sink = new Sink(this, entity);
27         unlock.addReceiver(sink);
28     }
29
30     public static void main(String[] args) {
31         doLoop(MM1Queue.class, args);
32     }
33 }
```

Listing 1: The M/M/1 implementation code showing a simple queuing system.

5 MODEL EXAMPLES

This Section illustrates two uses of the DES extension. We first describe a simple M/M/1 queue example to give a general idea of how a model may be implemented. Then, we demonstrate how we are using the MASON DES extension in a large supply chain model as a proof of application to real models.

5.1 M/M/1 Queue

Queuing systems simulate queues of entities arriving randomly and waiting in line to be processed and are widely used as they are common in real-life situations. An example would be a queue of airline passengers waiting to pass through a single security check. The M/M/1 queue example has an exponential inter-arrival time, exponential service time, one server, infinite capacity, and an infinite calling population.

Listing 1 shows the M/M/1 Queue implemented in MASON without visualization code. Like all MASON ABM models, this model subclasses from *SimState*, and the *start()* method prepares the simulation to run. We begin by creating a prototypical *Entity* called *foo*. We then create a *Source* process, which generates Entities of type foo. The Source is set to generate foo Entities at a rate chosen by an exponential distribution and then is scheduled to generate the first Entity at a random initial time. It will reschedule itself as needed to generate future Entities.

As the Source generates Entities, it offers its available Entities to a *Lock* process. The Lock will accept the Entity and pass it on only if it can allocate a single server, implemented as a *CountableResouce*, from a

Pool. If it cannot, the offer is rejected, and the Source holds onto its Entity to offer it later with others newly created. In this simple example, Source acts effectively both as a source of Entities and as a *Queue*. If we liked, we could have explicitly added a Queue process in front of the Lock.

If the Lock has allocated a CountableResource, it then passes its offered Entity forward, offering it in turn to a *Delay* process. The Delay has an infinite capacity, and so it always accepts the Entity, selecting a random delay time for the Entity under another Exponential distribution. The Delay automatically adds itself to the Schedule to wait this time. When the time is up, the Delay offers the Entity to an *Unlock* process. The Unlock always accepts Entities, forwards them on, and places one CountableResource back into the Pool for the Lock to retrieve.

When the Delay releases its Entity and the Unlock has released the CountableResource back to the Pool, how does the Source know to offer another Entity? Normally the Source offers existing Entities only when it creates new ones. However the Delay has set the Source to be its *slack provider*, and the Lock as its *slack receiver*. When the Delay releases its Entity, it has slack in its capacity, and so *after* offering the Entity through the Unlock (which frees the Lock), it goes to the Source and asks it to offer another Entity to the Lock. The Source then does this immediately.

Finally, the Unlock offers the Entity to a *Sink* process, the exit block, which accepts and throws the Entity away. The Lock, Delay, and Unlock collectively form the server portion of the M/M/1 Queue. Note that only the Source and Delay are scheduled, as they are the only processes that need to be pulsed at specific times. The Sink process simply accepts Entities offered. The remaining processes are filters that forward or reject Entities in zero time.

5.2 Supply Chain Model

Using our MASON DES extension, we have built a large model of a pharmaceutical supply chain to study approaches (using ABM and optimization) by which criminal networks may attack it, and how to mitigate against such attacks. We discuss the DES portion of this model at a high level to demonstrate how the MASON's DES classes can be easily customized and extended to represent production units modeling a realistic scenario. The supply chain model combines the sourcing, production, stocking, distribution, and sale of a pharmaceutical product. The model allows users to simulate disruption events to observe how the system responds in order to develop countermeasures or to test if existing precautions are appropriate.

The model customizes DES classes and methods to realize the industrial process. For example, we have a *Production* class modeling a process within a factory that takes several resources and generates a product. The class includes various DES components available in MASON: several Queues to store the resources, a Delay that models the production stage, and an additional Queue to control the throughput. We can also include another pair of Queues and Delays to shape transportation and testing stages. The resulting class allows the modeler to think of the process as a single abstract concept. This abstraction is important in complex production flowcharts where it is desirable to group the whole process into manageable blocks.

In the supply chain model, we are interested in simulating a scenario where several (notionally intentional) disruption events happen, damaging the supply chain. We use a *Disruption* class that contains a list of events with time, type, location, and magnitude of the disruption. We schedule each disruption event using this information at a specific simulation time. During a scheduled disruption, a portion of the supply chain behavior is modified (damaged) for a specific period of time. For example, a disruption may destroy some amount of a product in a storage facility, or it may make the product unavailable for some time. The consequences of a disruption event are easy to implement since the modeler only needs to override the DES component's methods, for example, letting it discard some elements. We model a variety of standard approaches taken to deal with both intentional and unintentional disruptions: particularly the use of safety

stocks. These are pools of products stored in abeyance at every stage along the supply chain and released during emergencies to keep the supply chain running.

This model heavily customizes both MASON and the DES extension components, and its large scope has served as a good testbed for assessing the efficiency of the toolkit, as well as flushing out bugs and identifying weaknesses in the design. In our research project, we expect to build multiple large supply chains to stress-test the system.

6 COMPARISON TO ANYLOGIC

AnyLogic (AnyLogic 2000) is a commercial HS software tool based on a visual development style that allows users to build simulations with little to no coding skills. These simulations can use ABM, DES, System Dynamics, or any combination of these three methods. Because of these characteristics, AnyLogic is one of the most used tools for HS development, as discussed in Section 2.3. For this reason, it is useful to describe certain ways in which AnyLogic and MASON's approaches differ.

MASON provides GUI visualization, inspection, and data analysis tools, but no graphical interface for development: the modeler must develop the simulation in Java. There are advantages to both approaches. In our experience, while AnyLogic's graphical editor assists novice developers, it also imposes limitations on the creation of complex simulations, in much the same way that NetLogo's graphical editor does. It leads the modeler to think in a particular way, hides many technical details an expert user might need to understand or modify, and can make customization difficult. Even when it is possible to create new components or change existing ones, doing so is challenging and discouraged. In AnyLogic, the user largely alters components by adding simple Java expressions at predefined points (hooks) in each construct called *actions*, which represent a particular moment of the simulation where the expression will be executed. Relying primarily on the graphical editor makes development quicker and easier compared to libraries like MASON, particularly for novices. But at the same time, this brings a tightly-coupled code structure where model execution cannot be separated from the visualization. AnyLogic was conceived as commercial software, so it is understandable that many low-level components and technical details are not explicitly described and are difficult to access.

AnyLogic allows the user to realize hybrid simulations in different ways: for example, agents and processes coexisting in the same environment, DES processes built into agents shaping their behavior, or agents acting as entities in a process. AnyLogic's graphical interface simplifies such model development, allowing the user to define agents and processes in the same editor, and agents within processes or processes within agents. The interaction between the components of different approaches applies the same logic to every AnyLogic simulation, namely (1) the ability to access and modify any object from any other object through function calls, (2) agent behavior shaped using state charts susceptible to changes to key variables, and (3) communication based on message-passing that allows triggering state changes.

MASON likewise allows for a variety of ABM-DES hybrid simulation combinations but adopts a different approach. Specifically, because they are implemented entirely using standard MASON objects and share MASON's schedule, DES processes have unfettered access to MASON fields and other facilities. Likewise, any MASON agent can access the DES processes as it sees fit. MASON can trivially support multiple DES flow graphs in parallel and even ones recursively contained inside agents (or vice versa).

7 CONCLUSION AND FUTURE WORK

We have introduced an extension to MASON ABM toolkit to enable DES model development. Our goal is to support researchers seeking to combine ABM and DES in integrated hybrid models with an open-source library. HS is a growing trend still in expansion but modelers must rely on custom solutions or commercial software to develop these models. In this field, AnyLogic is the primary tool thanks to its visual development

style, which provides ease of use and accessibility. However, its approach and commercial nature make AnyLogic restrictive for modelers wanting to realize more complex or higher-performance simulations or to integrate them with other tools.

Though we have used it successfully in large models representing a real-world scenario, MASON's DES extension is still in its infancy. In the coming months, we expect to make significant changes to its structure, naming, and organization. Currently, the system needs substantial improvements to its visualization tools. In addition to the *multi* and *macro* facilities, it needs more and better abstraction mechanisms to simplify the model design. And importantly, the system presently only has rudimentary statistics, a DES-critical feature that we need to rectify. A future direction for MASON's DES facility is the introduction of parallel computing, exploiting thread parallelism to enable the simultaneous execution of different DES processes or even embedded models. We will also examine porting the DES extension to Distributed MASON.

ACKNOWLEDGMENTS

The MASON DES extension was funded by National Science Foundation Grant 1727303. The DHS model was funded by the Department of Homeland Security under BOA number 70RSAT18G00000001, task order number 70RSAT21FR0000127. The content of this publication does not necessarily reflect the views or policies of the Department of Homeland Security, nor does mention of trade names, commercial products, or organizations imply endorsement by the U.S. Government.

REFERENCES

Antelmi, A., G. Cordasco, G. D'Ambrosio, D. De Vinco, and C. Spagnuolo. 2022. "Experimenting with Agent-Based Model Simulation Tools". *Applied Sciences* vol. 13, pp. 13.

AnyLogic 2000. "AnyLogic Simulation Software". https://www.anylogic.com/. The AnyLogic Company, Accessed Jan. 28, 2023.

Arena 2000. "Arena simulation software". https://www.rockwellautomation.com/en-us/products/software/arena-simulation.html. Rockwell Automation, Inc., Accessed Jan. 28, 2023.

Brailsford, S. C., S. M. Desai, and J. Viana. 2010. "Towards the holy grail: Combining system dynamics and discrete-event simulation in healthcare". In *Proceedings of the 2010 Winter Simulation Conference*, pp. 2293–2303.

Brailsford, S. C., T. Eldabi, M. Kunc, N. Mustafee, and A. F. Osorio. 2019. "Hybrid simulation modelling in operational research: A state-of-the-art review". *European Journal of Operational Research* vol. 278, pp. 721–737.

Chan, W. K. V., Y.-J. Son, and C. M. Macal. 2010. "Agent-based simulation tutorial — simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation". In *Proceedings of the 2010 Winter Simulation Conference*, pp. 135–150.

Cimini, C., G. Pezzotta, A. Lagorio, F. Pirola, and S. Cavalieri. 2021. "How can Hybrid Simulation support organizations in assessing COVID-19 containment measures?". *Healthcare (Basel)* vol. 9, pp. 1412.

Nick Collier 2001. "RePast: An Agent Based Modelling Toolkit for Java". http://repast.sourceforge.net. Accessed Jan. 28, 2023.

Cordasco, G., V. Scarano, and C. Spagnuolo. 2018. "Distributed MASON: A scalable distributed multi-agent simulation environment". *Simulation Modelling Practice and Theory* vol. 89, pp. 15–34.

Dagkakis, G., and C. Heavey. 2016. "A review of open source discrete event simulation software for operations research". *Journal of Simulation* vol. 10, pp. 193–206.

Eldabi, T. 2021. "Systemic Characteristics to Support Hybrid Simulation Modeling". In *2021 Winter Simulation Conference (WSC)*, pp. 1–10.

Farsi, M., J. A. Erkoyuncu, D. Steenstra, and R. Roy. 2019. "A modular hybrid simulation framework for complex manufacturing system design". *Simulation Modelling Practice and Theory* vol. 94, pp. 14–30.

Franceschini, R., P.-A. Bisgambiglia, L. Touraille, P. Bisgambiglia, and D. Hill. 2014. "A survey of modelling and simulation software frameworks using Discrete Event System Specification". In *2014 Imperial College Computing Student Workshop*, Volume 43.

Furian, N., D. Neubacher, S. Vössner, M. O'Sullivan, and C. Walker. 2014. "Towards holistic modeling and simulation of discrete event and individual based behavior". In *Proceedings of the European Simulation and Modelling Conference. Porto*, Volume 2.

Heath, S. K., S. C. Brailsford, A. Buss, and C. M. Macal. 2011. "Cross-paradigm simulation modeling: challenges and successes". In *Proceedings of the 2011 winter simulation conference (WSC)*, pp. 2783–2797. IEEE.

Hlupic, V. 1999. "Discrete-Event Simulation Software: What the Users Want". *SIMULATION* vol. 73, pp. 362–370.

Huanhuan, W., Z. Yuelin, and Z. Meilin. 2013. "A framework for integrating discrete event simulation with agent-based modeling". In *2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering*, Volume 3, pp. 176–180.

King, D. H., and H. S. Harrison. 2013. "Open-source simulation software JaamSim". In *2013 Winter Simulations Conference (WSC)*, pp. 2163–2171.

Langarudi, S. P., R. P. Sabie, B. Bahaddin, and A. G. Fernald. 2021. "A Literature Review of Hybrid System Dynamics and Agent-Based Modeling in a Produced Water Management Context". *Modelling* vol. 2, pp. 224–239.

Liu, H., X. Liu, and E. Lee. 2001. "Modeling distributed hybrid systems in Ptolemy II". In *Proceedings of the 2001 American Control Conference.*, Volume 6, pp. 4984–4985 vol.6.

Sean Luke 1998. "ECJ: A Java-based Evolutionary Computation Research System". https://cs.gmu.edu/~eclab/projects/ecj/. ECLab Evolutionary Computation Laboratory, Accessed Jan. 28, 2023.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. 2005. "MASON: A Multiagent Simulation Environment". *SIMULATION* vol. 81, pp. 517–527.

Macal, C. M., and M. J. North. 2010. "Tutorial on agent-based modelling and simulation". *Journal of Simulation* vol. 4, pp. 151–162.

Minar, N., R. Burkhart, C. Langton, and M. Askenazi. 1996. "The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations". Technical report, Santa Fe Institute.

Mittal, A., and C. C. Krejci. 2017. "A hybrid simulation modeling framework for regional food hubs". *Journal of Simulation*.

Klaus G. Müller and Tony Vignaux 2002. "SimPy". https://simpy.readthedocs.io/en/latest/. Accessed Jan. 28, 2023.

North, M. J., N. T. Collier, J. Ozik, E. R. Tatara, C. M. Macal, M. Bragen, and P. Sydelko. 2013. "Complex adaptive systems modeling with Repast Simphony". *Complex Adaptive Systems Modeling* vol. 1.

Nouman, A., A. Anagnostou, and S. J. Taylor. 2013. "Developing a Distributed Agent-Based and DES Simulation Using poRTIco and Repast". In *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*, pp. 97–104.

OMNeT++ 2001. "OMNeT++ Discrete Event Simulator". https://omnetpp.org/. OpenSim Ltd., Accessed Jan. 28, 2023.

Page, B., and W. Kreutzer. 2006. "Simulating discrete event systems with UML and JAVA". *Environmental Science and Pollution Research* vol. 13, pp. 441–441.

Quesnel, G., R. Duboz, and E. Ramat. 2009. "The Virtual Laboratory Environment — An operational framework for multi-modelling, simulation and analysis of complex dynamical systems". *Simulation Modelling Practice and Theory* vol. 17, pp. 641–653.

Rondini, A., F. Tornese, M. G. Gnoni, G. Pezzotta, and R. Pinto. 2017. "Hybrid simulation modelling as a supporting tool for sustainable product service systems: a critical analysis". *International Journal of Production Research* vol. 55, pp. 6932–6945.

SIMUL8 2003. "Simul8". www.simul8.com. Accessed Jan. 28, 2023.

Keith Sullivan and Mark Coletti and Sean Luke and Andrew Crooks 2010. "GeoMason: Geospatial Support for MASON". https://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/. Accessed Jan. 28, 2023.

Sumari, S., R. Ibrahim, N. H. Zakaria, and A. H. Ab Hamid. 2013. "Comparing three simulation model using taxonomy: System dynamic simulation, discrete event simulation and agent based simulation". *International Journal of Management Excellence* vol. 1, pp. 54–59.

Taylor, S. J. E., and S. Robinson. 2006. "So where to next? A survey of the future for discrete-event simulation". *Journal of Simulation* vol. 1 (1), pp. 1–6.

Vempiliyath, T., M. Thakur, and V. Hargaden. 2021. "Development of a Hybrid Simulation Framework for the Production Planning Process in the Atlantic Salmon Supply Chain". *Agriculture* vol. 11, pp. 907.

Viana, J., T. B. Simonsen, F. A. Dahl, and K. Flo. 2018. "A Hybrid Discrete Event Agent Based Overdue Pregnancy Outpatient Clinic Simulation Model". In *2018 Winter Simulation Conference (WSC)*, pp. 1488–1499.

Vieira, A., L. Dias, M. Santos, G. Pereira, and J. Oliveira. 2019. "A ranking of the most known freeware and open source discrete-event simulation tools". In *Proceedings of the 31st European Modeling & Simulation Symposium (EMSS 2019)*, pp. 200–2019.

Wang, B., S. Brême, and Y. B. Moon. 2014. "Hybrid modeling and simulation for complementing Lifecycle Assessment". *Computers & Industrial Engineering* vol. 69, pp. 77–88.

Uri Wilensky 1999. "NetLogo simulation platform". http://ccl.northwestern.edu/netlogo/. Accessed Jan. 28, 2023.

Zhang, B., W. K. Chan, and S. V. Ukkusuri. 2011. "Agent-based discrete-event hybrid space modeling approach for transportation evacuation simulation". In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pp. 199–209.

AUTHOR BIOGRAPHIES

GIUSEPPE D'AMBROSIO is a PhD student in the ISISLab laboratory at the Università degli Studi di Salerno. His research interests include reliability and scalability of scientific computing applications, exploiting advanced computational paradigms including cloud computing, parallel methods, and distributed computing.

SEAN LUKE is a Professor in the Department of Computer Science at George Mason University, Fairfax, Virginia. He has interests in stochastic optimization, multiagent systems and multiagent learning, agent-based modeling, swarm robotics, and computational creativity.