An Introduction to Discrete-Event Modeling and Simulation

Oliver Ullrich[1*], Daniel Lückerath[2]

[1]School of Computing and Information Sciences, Florida International University, ECS 243C, 11200 SW 8th St, Miami FL-33199, *oullrich@cs.fiu.edu

[2]Institut für Informatik, Universität zu Köln, Weyertal 121, 50931 Köln, Germany

SNE 27(1), 2017, 9-16, DOI: 10.11128/sne.27.on.10362
Received: October 10, 2016
Accepted: December 10, 2016
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. Especially suitable for the modeling and simulation of technical systems in a wider sense, discrete-event simulation is one of the most important and most versatile tools of the craft. Both students and practitioners should be familiar with its principles and mechanics, in particular if they are interested to look beyond their favorite modeling tool's GUI.
This paper presents a short tutorial on modeling and simulation techniques, with a focus on discrete-event simulation. After an introduction to the subject matter, some event-oriented modeling fundamentals are discussed, followed by a description of simulation execution principles. A slim software framework is introduced aimed at simplifying model building and evaluation, followed by the presentation of a small sample of recently completed discrete-event simulation studies. The paper concludes with a short summary of the lessons learned and some pointers to additional literature.

Introduction

Decades after its inception (see [19]), the discrete-event modeling and simulation method is still a mainstay of the trade, with hundreds of discrete-event simulation models being built, executed, and described (for a small selection of recently completed studies see section 4).

Discrete-event based methods are especially suitable if the examined system's behavior can be described as a set of stochastically influenced, connected components, changing their states at discrete points in time. A discrete-event simulation model represents such a system as a set of connected, attributed entities, executing, and communicating by, events and activities.

Common applications range from communications to transportation, and especially cover the simulation of technical systems in a wider sense. It is less suitable in areas like astronomy, meteorology, or materials sciences, while social scientists often prefer agent-based modeling for their simulation experiments.

While discrete-event simulation is easily understood, it is also very expressive and powerful. Single researchers, students, or practitioners can create high-validity models and evaluate their behavior. Discrete-event simulation includes three worldviews: activity-oriented, process-oriented, and event-oriented simulation (for detailed information see e.g. [5], pp. 88-108). While these views each emphasize different model components to represent the investigated system, they all use the same basic modeling principles and can easily be translated into another.

This paper presents an introduction to modeling and simulation techniques, with a focus on discrete-event simulation. It is especially addressed to students of the craft, and to practitioners who might want to look beyond their usual modeling tool's GUI. In addition to describing the modeling process, the execution of simulation models, and a number of typical academic and real-world applications, the paper also includes a description of an open-source Java framework which encapsulates and hides the event-oriented method's technical details, and provides a slim API for modelers to start implementing a model without having to be concerned with the underlying mechanics. The framework can be used (and has been used) for both simple experiments and full blown case studies.

The paper continues with an introduction of basic modeling methods with a focus on event-oriented concepts (section 1), including entities, events, activities, as well as input distributions and their modeling, followed

by a description of fundamental techniques of model execution (section 2), both for traditional event-oriented simulation and its object-oriented extension. Building up on this, the paper then introduces a slim event-oriented simulation framework (section 3) designed to allow the user to concentrate on modeling aspects and to isolate them from technical housekeeping work. The penultimate section (section 4) presents a sample of current or recently concluded case studies in a wide variety of application fields, including communications, disaster mitigation, health care, logistics, supply chain management, and transportation. The paper concludes with a short summary of the lessons learned and some recommendations for further reading (section 5).

1 Event-oriented Modeling

The modeling process starts with the examination and analysis of the real-world system to be modeled. It is decomposed into interacting components which perform processes, execute and generate events and interact by passing messages. The decision has to be made what components and processes should be part of the model, what should be parametrized, and what should be left out completely. Generally, if an entity or a process has a strong influence on the core behavior of the model, it should be incorporated. If the influence is weak, it might be wiser to parametrize it to avoid creating overcomplexity.

Discrete-event modeling is suitable for systems that can be described as a set of interrelated entities which only change their state (and subsequently the system's state) at discrete points in time as a result of their own behavior or of the behavior of other entities. Entities describe components of the system under investigation which have to be modeled explicitly in order to represent the system behavior relevant to the simulation study. Each entity possesses individual attributes whose values describe their current state and which affect their behavior. The potential behavior of an entity is modeled as a set of activities, i.e. a sequence of actions during some time period which may result in state changes of the acting entity as well as other entities of the model. The point in simulation time at which such a state change occurs is called an event. Subsequently, activities are always framed by events. The precise relationship between specific events and activities has to be defined by the modeler based on the goals of the simulation study. In particular, this includes the specification of the durations of activities, which can be modeled as deterministic as well as based on stochastically influenced parameters. The latter case is usually modeled using drawings from random distributions, whose type and parameters are acquired by analyzing relevant input data (e.g. using Microsoft Excel).

In discrete modeling some distribution types (see figure 1) are especially useful: Independent machine processing times, e.g. for the tooling of a component, can often be approximated by applying a normal distribution $\aleph(\mu, \sigma^2)$ with an average of μ and a standard deviation σ, or by using a log-normal distribution $\ln \aleph(\mu, \sigma^2)$. Arrival rates, e.g. of cars entering a freeway segment, are usually modeled using a discrete Poisson distribution $\text{pois}(\lambda)$ with an average number λ of events per interval. The time between two independent arrival events is often approximated with an exponential distribution $\exp(\mu)$ with an average interarrival time of $\mu = 1/\lambda$, λ again being the average number of events per interval (see [5], pg. 248).

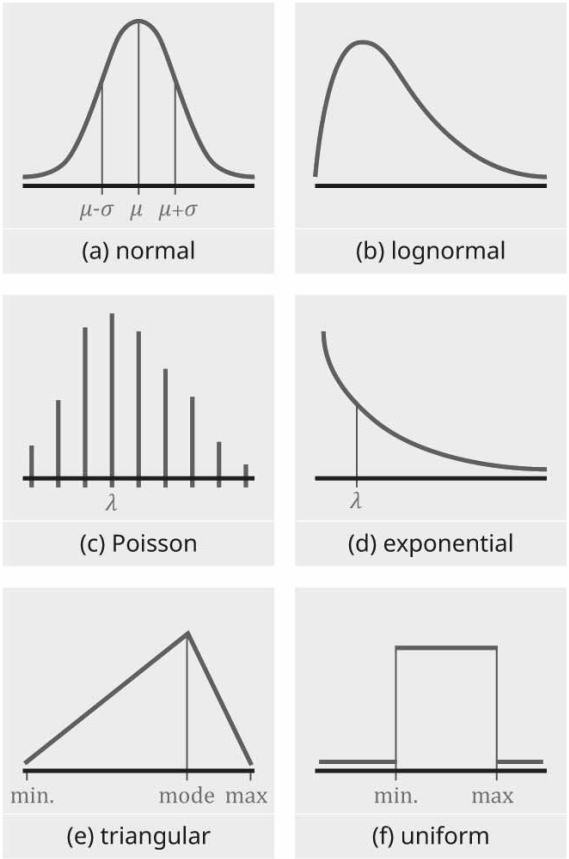

Figure 1: Distribution types.

If the available data is insufficient to determine a distribution's type and parameters, insight might be gained by interviewing local domain experts. If an expert is confident that a stochastically influenced process takes a time "usually in the vicinity of *mode* units, never less that *min*, never more than *max* units," the process duration can be modeled as a drawing from a triangular distribution tri$(mode, min, max)$ with the parameters *min*, *max*, and *mode*. If available expert knowledge is as imprecise as "somewhere between *min* and *max*," the modeler has to make due with a uniform distribution U(min, max) with the parameters *min* and *max*.

Because the model state only changes at the occurrence of events, it follows that it stays unchanged between two subsequent events and that in general an activity can't be influenced by an event which occurred after the activity has already started.[1] The event-oriented worldview takes advantage of this by representing the passage of time in the real-world system using variable time steps for the incrementation of simulation time, i.e. the simulation time jumps from the occurrence of one event to the occurrence of the next event. The event-oriented simulation process can thus be understood as a sequence of snapshots of the model state, starting with the earliest snapshot (e.g. at simulation time $t_s = 0$) and ending once no more snapshots are generated. The first snapshot in the sequence has a special status: It has to be defined by the modeler and is used to initialize the states of all model entities. Furthermore, it is used to schedule exogenous events, i.e. events whose origins lie outside the boundaries of the model (in contrast to endogenous events, whose origins lie inside the boundaries of the model). Often, the practitioner's goal is not to examine a model's regular behavior, but to evaluate its reaction to specific disruptions or disturbances (e.g. the blockade of a track segment or a machine failure). In such a case one does not want to rely on a disruption occurring randomly, especially if these are very rare. Therefore, disruptions are systematically injected into the model as exogenous events with a specific time stamp during the first simulation snapshot. When the simulation is started, it executes the routine events, until it reaches the exogenous event. The model's state is changed accordingly and the model logic designed to cope with a disruption springs into action; its effectiveness can now be examined systematically.

[1] It is possible to deviate from this assumption in individual cases, if the simulation framework used to implement the model provides methods and data structures for the rescheduling of already scheduled events, thus possibly aborting an activity that has already started.

A commonly used tool to plan and visualize event-oriented models is the event activity chain (see figure 2). Here, events are represented as hexagons, framing activities which are represented as round cornered rectangles. Conditions to be fulfilled, or a decision on what path to follow, can be represented by *or* ($\vee$), *exclusive or (xor)*, or *and* ($\wedge$) connections. The chain's elements can further be attributed by distributions and other parameters.

2 Event-oriented Model Execution

Once the components of a simulation model are identified, including their attributes, relations, activities, and events, the model is implemented as software and subsequently executed. Using object-oriented programming languages, model entities can be implemented one to one as objects, with entity types being realized as classes.

The data structure used to represent events has to include fields for: the event type, e.g. START_SIMULATION, BEGIN_PASSENGER_EXCHANGE, RELEASE_WORKSTATION; the time stamp, usually stored as an integer value, with one increment constituting the smallest representable time increment, e.g. one second or one millisecond; and additional information depending on the model.

These simulation events are managed by the Future Event List (FEL). This data structure is usually implemented as a priority queue, an abstract data type which orders elements according to a key, in this case to the time stamp. Amongst other functions, the FEL provides interfaces to find and remove the element with the smallest key, insert new elements at the right position in the FEL, and test for emptiness. Usually, the dequeuing order of events with identical time stamps is not guaranteed, therefore making it more difficult to represent concurrent access to shared resources. If such functionality is desired, the events have to include an additional attribute, e.g. a general index, which can be used as secondary sorting key.

During the simulation run the simulation engine repeatedly retrieves (and thus removes) the event with the minimum time time stamp value from the FEL and executes it. In the course of the execution the simulation time is set to the currently executed event's time stamp, the model's state variables are updated, and follow-up events are generated

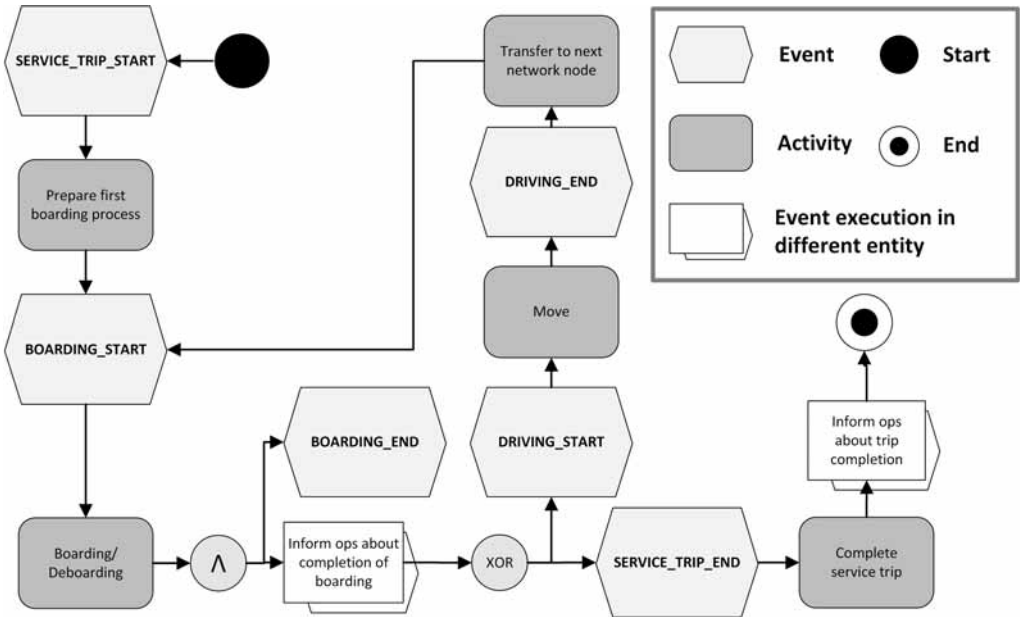

Figure 2: Example for event-activity-chain notation depicting a simplified model of a transit vehicle executing a service trip.

and inserted into the FEL (e.g. during the execution of BEGIN_PASSENGER_EXCHANGE an event END_PASSENGER_EXCHANGE might be scheduled 23 seconds into the model's future). The simulation therefore jumps from time stamp to time stamp, and ignores the time in between two subsequent events. Stochastic elements, e.g. processing durations or interarrival times, are drawn from random distributions, usually by utilizing a library providing a large number of distribution types. How to efficiently generate these distributions, and how to ensure their unpredictability and randomness, is a research area by itself (see [10]), and is well beyond the scope of this introduction.

Using the described data structures and concepts, the simulation run is executed by iterating through a simple loop (see figure 3): (1) Start the run, set the simulation time to zero, send a *START_SIMULATION* event when appropriate; (2) get the event e_i with the minimum time stamp from the FEL, set the simulation time to e_i's time stamp; (3) execute event e_i according to its type, thereby updating the model state and enqueuing follow-up events; (4) check the stop conditions, which usually include at least a check whether the FEL is empty. If the stop conditions are fulfilled, end the simulation run by continuing with step (5). If not, repeat the loop from step (2) by dequeueing the next event e_{i+1}. (5) Prepare statistics and evaluate results.

In the object-oriented extension (see figure 4) the event data structure additionally contains a reference to the source entity and a list of entities which are the event's addressees. The central event handling routine is replaced by a dispatcher which calls the addressees' event handling routines. If one event includes more than one addressee, the order of these calls usually cannot be guaranteed.

With potentially hundreds or thousands of events in the queue at the same time, it is obvious that the performance of the priority queue implementation is an important factor in the method's overall performance. A number of data structure is commonly used to implement the priority queue, including hashing structures, heaps, binary trees, or (in small models) linked lists. For a more detailed discussion of priority queue data structures see [16].

3 An Event-oriented Simulation Framework

Surfside[2] is a slim Java framework for event-oriented simulation. It has been used in a number of applications (see [12], [13], [15], [25]), mainly in the area of computational transportation science. *Surfside* is designed to

[2]available under an open source license at www.oullr.de/surfside

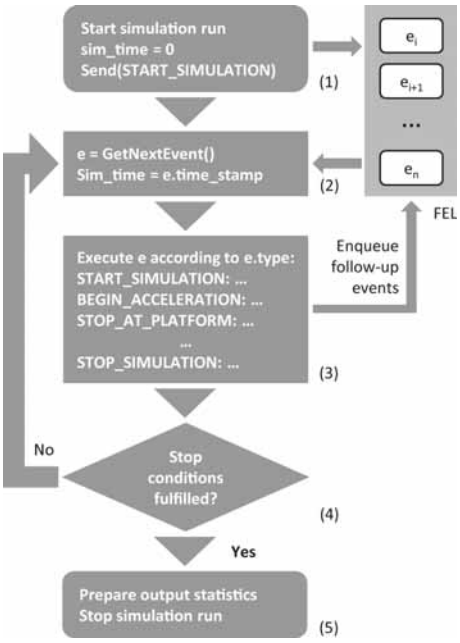

Figure 3: Basic event-oriented simulation loop.

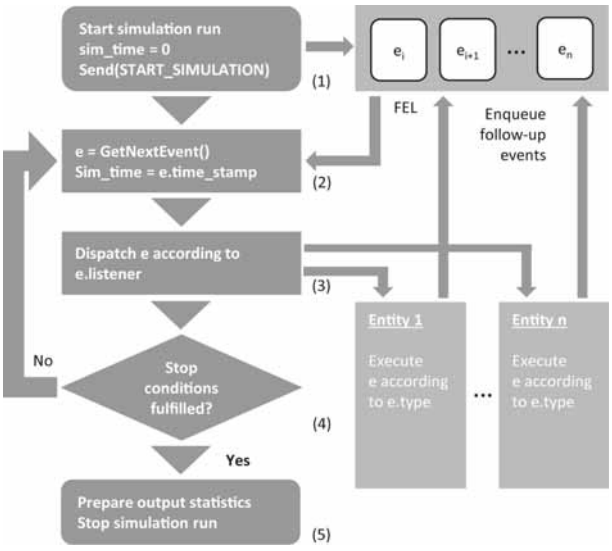

Figure 4: Object-oriented extension of the event-oriented simulation loop.

be easily understood, and to help modelers to concentrate on the actual modeling task by isolating them from technicalities.

Simulation events are represented by instances of the class *Event*. Each instance includes an event type value, a time stamp, a reference to the event's generating entity, and a list of event listeners or addressees, i.e. objects implementing the interface *EventHandler*. The class *Event* can be extended to include further application-specific data.

Event types are administrated by the class *EventTypeManager*, which prevents dublicated event type values and provides centralized access to all registered event types. A set of standard event types, defined in the class *EventType*, is added to the *EventTypeManager* by default. To use custom event types the user has to create her own event types and add them to the *EventTypeManager*.

Entity types are realized as classes implementing the interface *EventHandler*. This interface demands a method *handleEvent(Event)* which consumes simulation events addressed to the entity, evaluates event data, changes the model state, and generates follow-up events by calling *Simulation.addEvent(Event)*. The interface also demands a method *reset()* to reset the entity's state in between simulation runs, and some other minor methods.

The abstract class *Simulation* encapsulates the functionality to prepare, execute, and evaluate a number of simulation runs and is usually extended to be the simulation application's main class. It is implemented as a Singleton and itself implements the *EventHandler* interface, and can thus function as a conceptual model entity. If all simulation events are handled by the class extending *Simulation*, the model followed the classic event-oriented paradigm without its object-oriented extension.

The user's code takes control in an extended constructor, utilizing calls to *addEntity(EventHandler ...)* to add one or more entities to the model. The number of simulation runs to be executed is set via *setNumberOfRuns(int)*. The user's code then calls *start()*, thereby commencing the execution of the given number of simulation runs. The user also has to implement the abstract methods *prepareNextRun()*, *eventPostProcessing(Event)* and *evaluateRun()*, which are used to reset the model in between simulation runs, evaluate and collect data after processing a single event, and to evaluate data collected after finishing a run, respectively. At the start of each simulation run, an event of type *SIMULATION_START* is automatically generated and sent to the class extending Simulation, at the conclusion of each run, if the user did not stop the simulation herself, an event of type *SIMULATION_END* is automatically sent. The class also manages the FEL, to which events can be added by using *addEvent(Event)*.

Specific events can be removed from the list by calling *removeEvent(Event)*. The framework automatically manages the FEL mechanics encapsulated in the class *FutureEventList*. This class can be extended, thereby allowing experiments with FEL data structures themselves.

Surfside provides a number of helper classes for input distribution generation, output logging, etc: The class *Distribution* encapsulates parametrized random distributions for single time periods with homogeneous parameters, supporting uniform, triangular, normal, lognormal, exponential, and poisson distributions. The class *CompositeDistribution* encapsulates functionality to model processes whose distributions and parameters change over time, by allowing the assemblage of mutlipe *Distribution* instances with inhomogeneous parameters and different time periods. The class *PropertiesManager* administrates the simulation parameters and offers functionality to read parameters from the command line as well as from XML files. The classes *Column* and *Exporter* help to collect and write output data. The latter is an interface defining the minimum functionality for storing simulation data, e.g. *add(Column)* and *log()*, which should be implemented by custom data export classes. A simple implementation for writing data to a Microsoft Excel spreadsheet based on a user defined template using the JExcelAPI libary is already provided by the class *JExcelAPIExporter*.

4 Current Applications

Presented here is a small sample of current or recently concluded discrete-event simulation case studies in a wide variety of application fields, collected to give an impression of the method's flexibility and utility. Some applications show small, very specific models, constructed and implemented by single practitioners or a small team, while others consist of large multi-person projects.

Communications: Artuso and Christiansen (see [3]) examine inter-cell interferences in long term evolution (LTE) cellular networks. As it is very costly to assess possible solutions in the field, network operators are interested in the evaluation of potential remedies in simulated scenarios. Artuso and Christiansen find that, by applying certain techniques, improvements from 25% to 40% in relation to the baseline scenario can be achieved. Cetinkaya et al. (see [6]) analyze the effects of perturbations on resilient communication networks. They use discrete-event simulation to inject disturbances into a representation of the USA's Sprint backbone network, thus assessing impacts on performance and dependability.

Disaster mitigation: Kuchel (see [11]) examines the impacts of civil unrest and violence on large-scale evacuation operations. He provides a model to analyze and evaluate scenarios with the goal of reducing evacuation time, and to identify limiting factors and choke points for given local transportation networks. Suk Na and Banerjee (see [23]) combine agent-based and discrete-event simulation to provide a decision support system for network evacuation strategies in case of natural disasters. Their model includes an embedded geographic information system, and allows to evaluate possible shelter positions, vehicle types, and evacuation priorities. They examine a number of scenarios based on an assumed natural disaster in Galveston, Texas.

Health care: Allen et al. (see [2]) examine the rescheduling of medical surgery schedules as a consequence of disruptions during the operational day. Their discrete-event model focuses on the impacts of rescheduling measures on hospital personnel, especially on further delays resulting from delayed communications regarding the updated schedule. Mayorga et al. (see [18]) construct a simulation model to assess the health and economic impacts of various smoking cessation techniques. Their model represents 100,000 smokers receiving one of five alternative smoking cessations interventions, including several medications, nicotine replacement therapy, and counseling. The results indicate that while all treatments have a positive average effect on the participants' quality of life, a small long-term preference for a certain pharmaceutical treatment can be identified. Pooya et al. (see [20]) use discrete-event simulation to analyze and assess the reliability and safety of the radiation therapy care delivery process, focusing on possible optimization by quality assurance protocols. They find that "incidents" caused by mistakes in the organ delineation and dose calculation steps are most costly, and can be reduced by improved quality assurance procedures.

Logistics and supply chain management: Abbot and Marinov (see [1]) examine alternative designs for railroad interchange yards, which facilitate the exchange of rolling stock between Britain's planned "High Speed Two" railway and conventional networks. Their model consists of a number of sub-models, representing e.g. locomotive change, brake inspection, and the train ar-

rival process. They find that under the given yards designs high-speed and conventional rolling stock can be operated together, but at a speed penalty. Iannone et al. (see [9]) assess the impact of complex management decisions on planning and operations of inter-modal logistical terminals. Based on data from Italian terminals, they model both scheduling and loading/unloading processes of cargo on large marine vessels, finding that by adopting a different scheduling method a two digit percentage of monetary cost can be saved. Leonard et al. (see [14]) analyze the launch process of the NASA Space Launch System, focusing on the supply of the four main liquid launch commodities: hydrogen, oxygen, nitrogen, and helium. Their study confirms that, applying a new refilling strategy, the turnaround time could be reduced from over seven days to 48 hours. Schroer et al. (see [22]) model a number of terminals of the port of Rotterdam, Netherlands. They investigate options for the transport of cargo containers between these sub-systems, and consider several discussed configurations of the port's future extensions.

Transit: Ullrich et al. (see [24]) use a discrete-event simulation model of light-rail transit to examine the conditions under which the application of regular timetables reduces service delays resulting from inevitable small disturbances. They find that for regular timetables to have an impact, the light-rail system has to adhere to three conditions: resources like tracks and splits have to be shared, the traversal time variance has to be comparatively low, and resources cannot be redundant. Wales and Marinov (see [26]) describe delayed service issues in the British Tyne and Wear Metro light-rail system. They develop a simulation model to analyze the origins of these delays, and to assess a number of potential mitigation techniques. They find that by applying these techniques, the system's punctuality could be improved by 15% to 17% in relation to the current state.

A collection of further recent use cases, especially in the areas of automotive and manufacturing simulation, can be found in [4].

5 Conclusions

This paper presented an introduction to discrete-event modeling and simulation, in particular to the event-oriented worldview. It shared some fundamental modeling techniques, described the execution of simulation models, and introduced *Surfside*, a slim framework for event-oriented simulation. An overview of some current case studies showed the relevance of the paradigm for researchers and practitioners.

Discrete-event modeling and simulation is a very simple, easy to understand, but also very powerful approach. It is especially useful for modeling technical systems in the wider sense, including communications, disaster mitigation, health care, logistics, supply management, and transportation. With the help of slim frameworks a single researcher or practitioner can build, execute, and evaluate a small but meaningful discrete-event model.

For further, more detailed research a number of books and articles can be recommended: Banks et al. (see [5]) is a seminal work on discrete-event modeling and simulation. Fishman (see [7]) is an excellent alternative, especially for readers with an interest in operations research. Model verification and validation are the single most important tasks in any simulation project. Sargent (see [21]) describes a set of paradigms and techniques to ensure a model's high validity and credibility. Even 15 years after its release, Fujimoto (see [8]) is still the weapon of choice for those who want to understand the fundamentals of parallel and distributed simulation. For modelers who desire to take a step beyond discrete event mechanics, Macal and North's introduction to agent-based modeling and simulation (see [17]) can be recommended.

6 Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Nos. I/UCRC IIP-1338922, AIR IIP-1237818, III-Large IIS-1213026, MRI CNS-1532061, MRI CNS-1429345, MRI CNS-0821345, MRI CNS-1126619, MRI CNS-0959985, RAPID CNS-1507611, and U.S. DOT Grant ARI73.

References

[1] Abbott D, Marinov MV. An event based simulation model to evaluate the design of a rail interchange yard, which provides service to high speed and conventional railways. In: Simulation Modelling Practice and Theory, Vol. 52, 2015, pp. 15-39.

[2] Allen RW, Taaffe KM, Ritchie G. Surgery Rescheduling Using Discrete Event Simulation: A Case Study. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 1365-1376.

[3] Artuso M, Lehrmann Christiansen H. Modeling and Event-Driven Simulation of Coordinated Multi-Point Joint Transmission in LTE-Advanced with Constrained Backhaul. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 3131-3142.

[4] Bangsow S. (Ed.): Use Cases of Discrete Event Simulation. Springer, 2012.

[5] Banks J, Carson JS, Nelson BL, Nicol DM. Discrete-Event System Simulation, 5th edition. Pearson, 2010.

[6] Cetinkaya EK, Broyles D, Dandekar A, Srinivasan S, Sterbenz JPG. Modeling communication network challenges for Future Internet resilience, survivability, and disruption tolerance: a simulation-based approach. In: Telecommunication Systems, Vol. 52, 2013, pp. 751-766.

[7] Fishman GS. Discrete-Event Simulation: Modeling, Programming, and Analysis. Springer, 2001.

[8] Fujimoto R. Parallel and Distributed Simulation Systems. John Wiley & Son, 2000.

[9] Iannone R, Miranda S, Prisco L, Riemma S, Sarno D. Proposal for a flexible discrete event simulation model for assessing the daily operation decisions in a Ro-Ro terminal. In: Simulation Modelling Practice and Theory, Vol. 61, 2015, pp. 28-46.

[10] Knuth D. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Edition. Addison-Wesley, 1997.

[11] Kuchel D. Analyzing Noncombatant Evacuation Operations Using Discrete Event Simulation. In: Proceedings of 2013 Winter Simulation Conference, 2013, pp. 2751-2761.

[12] Kuckertz P, Randerath H. A model for airline personnel schedule simulation. In: Proceedings of ASIM-Workshop STS/GMMS 2014, ARGESIM Report 42, ASIM Mitteilung AM 149, ARGESIM/ASIM Pub., TU Vienna/Austria, Februar 2014, pp. 171-180.

[13] Kuckertz P, Ullrich O, Randerath B. A simulation based approach on robust airline job pairing. In: Simulation Notes Europe (SNE), Volume 22, Number 2, 2012, pp. 77-82.

[14] Leonard D, Parsons J, Cates G. Using discrete event simulation to model fluid commodity use by the space launch system. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 2954-2965.

[15] Lueckerath D. Ein Simulationsmodell fuer Oeffentlichen Personennahverkehr mit regelbasiertem Verkehrsmanagement. Dissertation, Univ. Koeln, 2016.

[16] Lueckemeyer G. A Traffic Simulation System Increasing the Efficiency of Schedule Design for Public Transport Systems Based on Scarce Data. Dissertation, Univ. Koeln, 2007.

[17] Macal CM, North MJ. Tutorial on agent-based modeling and simulation. In: Journal of Simulation, Vol. 4, 2010, pp. 151-162.

[18] Mayorga ME, Wheeler SB, Reifsnider OS, Kohler RE. A Discrete Event Simulation Model to Estimate Population Level Health and Economic Impacts of Smoking Cessation Interventions. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 1257-1268.

[19] Nance RE. A history of discrete event simulation programming languages. In: Proceedings of the 2nd ACM SIGPLAN conference on History of programming languages, 1993, pp. 149-175.

[20] Pooya P, Ivy J, Mazur L, Deschesne K, Mosaly P, Tracton G. Assessing the Reliability of the Radiation Therapy Care Delivery Process Using Discrete Event Simulation. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 1233-1244.

[21] Sargent RG. Verification and validation of simulation models. In: Journal of Simulation, Vol. 7, No. 1., 2013, pp. 12-24.

[22] Schroer HJL, Corman F, Duinkerken MB, Negenborn RR, Lodewijks G. Evaluation of inter terminal transport configurations at Rotterdam Maasvlakte using discrete event simulation. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 1771-1782.

[23] Suk Na H, Banerjee A. An Agent-Based Discrete Event Simulation Approach for Modeling Large-Scale Disaster Evacuation Networks. In: Proceedings of the 2014 Winter Simulation Conference, 2014, pp. 1516-1526.

[24] Ullrich O, Lueckerath D, Speckenmeyer E. Do regular time tables help to reduce delays in tram networks? - It depends! In: Public Transport, Volume 8, Issue 1, 2016, pp. 39-56.

[25] Ullrich O, Azzouzi C, Brandt B, Lueckerath D, Rishe N. Modeling Garage Parking. In: Simulation Notes Europe (SNE), Volume 26, Number 1, pp. 1-8.

[26] Wales J, Marinov M. Analysis of delays and delay mitigation on a metropolitan rail network using event based simulation. In: Simulation Modelling Practice and Theory, Vol. 52, 2015, pp. 52-77.