Mixed Discrete-Continuous Simulation for Digital Twins

Neha Karanjkar[1][a] and Subodh M. Joshi[2][b]
[1]Indian Institute of Technology Goa, India
[2]Department of Computational and Data Sciences, Indian Institute of Science, Bangalore, India

Keywords: Digital Twins, Mixed Discrete-Continuous Simulation, Python, SimPy.

Abstract: The advent of Digital Twins has precipitated the need for an open and flexible simulation framework with unique design considerations. A key requirement of such a framework is the ability to simulate systems containing a mix of discrete and continuous processes that may interact with each other. In this paper, we propose a framework for mixed discrete-continuous simulations particularly targeted for Digital Twin applications. The framework is based on SimPy, a popular discrete-event simulation library in Python. We first present a systematic approach by which continuous process simulations can be integrated within the event-stepped engine of SimPy. We illustrate the approach using an example. We then discuss the features of the proposed framework and the roadmap for its implementation.

1 INTRODUCTION

The emergence of Digital Twins is set to transform manufacturing, healthcare, urban planning, transportation and many other sectors by aiding monitoring and real-time decision making using Internet of Things (IoT) technology and real-time analytics. A Digital Twin refers to a digital representation (computer model) of the real system that is continuously kept in sync with the real system using periodic sensing of its health parameters and incorporates analytics for prediction, optimization and control of the real system. It is to be noted that the term ‘Digital Twin’ implies a two-way flow of information between the real entity and its digital representation (viz. sensing and control) and is distinct from a ‘digital model’ or a ‘digital shadow’ which generally imply a one-way information flow from the real system to its model. A taxonomy of Digital Twins and a detailed discussion of their engineering aspects are presented in (Fuller et al., 2020).

Simulation is a key aspect of Digital Twins. A summary of challenges and desired capabilities associated with the simulation of Digital Twins is presented in (Shao et al., 2019). The design of a simulation framework for Digital Twins is driven by the characteristics of the system to be modeled. While some system models may necessitate a continuous simulation framework (Aversano et al., 2020; Molinaro et al., 2021), a discrete-event simulation might suffice for other kinds of Digital Twins (Agalianos et al., 2020).

The focus of this paper is on open simulation frameworks for creating Digital Twins in application areas such as manufacturing and process control. In these applications, the phenomena to be modeled are often a *mix of discrete and continuous processes*. Thus the ability to perform mixed discrete-continuous (MDC) simulations is a key requirement for the framework. The other requirements for such a simulation framework are as follows:

1. The ability to model heterogeneous systems containing different kinds of continuous processes, each possibly requiring a different numerical method for its solution and/or different characteristic time-step sizes.
2. Support for capturing the effect of periodic sensor updates from the real system on the model’s state.
3. The ability to perform real-time simulation.
4. The framework should be open-source and flexible. It should be easy to integrate existing libraries for enabling analytics and visualization (e.g. optimization, machine learning, data handling, scientific computing and plotting libraries) into the framework.
5. The language used by the framework should sup-

[a] https://orcid.org/0000-0003-3111-1435
[b] https://orcid.org/0000-0002-9239-8866

Karanjkar, N. and Joshi, S.
Mixed Discrete-Continuous Simulation for Digital Twins.
DOI: 10.5220/0010580804220429
In *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2021)*, pages 422-429
ISBN: 978-989-758-528-9

port modular descriptions and the use of object oriented features for modeling complex systems with many interconnected components.

While there exist a number of frameworks that are targeted separately for either continuous simulation or discrete-event simulations, the requirement of simulating both discrete and continuous processes together, possibly interacting with each other, introduces some challenges. Section 2 presents a brief review of approaches and existing frameworks for continuous, discrete and hybrid simulation. A majority of the existing frameworks for MDC simulations are either commercial or domain-specific. OpenModelica is a notable exception which offers an open simulation environment based on a declarative modeling language (Modelica) and a library of component models for multi-domain simulation (Fritzson et al., 2020).

However, there still exists the need for a mixed simulation framework that is written in a general-purpose, object-oriented language which allows integration with existing continuous simulation frameworks. Python is an attractive choice for implementing such a framework because of its wide user base, ease of use and the availability of a number of libraries for analytics and visualization. In this paper, we present the outline for an open-source, Python-based mixed simulation framework motivated by the requirements listed above.

The proposed framework is based on SimPy, a process-based discrete-event simulation library in Python (SimPy-Team, 2020). Processes in SimPy are implemented using Python's generator functions and can be used to model active components. The processes are managed by an *environment* class, which performs time advancement in an event-stepped manner using a global event queue. The system to be modeled can be described in Python using a few SimPy constructs and does not require the user to learn a new modeling language. SimPy also supports real-time simulation. However, SimPy is designed for discrete-event simulation and currently offers no features for modeling continuous systems (SimPy-Team, 2020).

In this paper we present a systematic approach for integrating continuous models into the event-stepped simulation engine of SimPy for MDC simulations. The framework is particularly suited for systems where the continuous entities are few and loosely coupled and their interactions can be modeled via events. We present the framework, illustrating its key ideas using a simple example, and describe the roadmap for its implementation.

The rest of the paper is organized as follows: in the next section, we present a detailed review of existing approaches to continuous, discrete-event and mixed simulations. In Section 3 we present some design considerations for simulation of MDC systems and describe the proposed framework. The roadmap for implementation is presented in Section 4.

2 A REVIEW OF SIMULATION APPROACHES AND FRAMEWORKS

Depending on the type of the system to be modeled, simulation approaches can be broadly classified into the following categories:

A. Continuous System Simulation.
Continuous systems are characterized by continuous evolution of the state variables. A few examples of such systems include transient heat conduction in solids, fluid flows, acoustic wave propagation etc. Under certain conditions, fundamentally discrete systems such as highway vehicle traffic can be modeled as continuous processes, e.g. evolution of vehicle density over time (Lighthill and Whitham, 1955; Sreekumar et al., 2019). The mathematical models used for describing the dynamics of such systems often take the form of ordinary/partial differential equations (O/PDE) or mixed differential-algebraic equations (DAE). Simulating continuous systems thus involves solving the differential equations using suitable numerical schemes (such as the finite element spatial discretization method) as well as the appropriate numerical time integrators. Time is advanced in a regular manner using a step-size that is either fixed, or adjusted dynamically over the course of a simulation. A detailed description of continuous processes and their simulation aspects can be found in (LeVeque, 1990; Cellier and Kofman, 2006) and the references therein. In practice, frameworks such as FEniCS (Logg and Wells, 2010), Deal II (Bangerth et al., 2016), OpenFOAM (Weller et al., 1998) are used for continuous multiphysics simulations of complex systems. Numerical techniques such as reduced order models (ROM) (Chinesta et al., 2011; Feng, 2005) and Machine-Learning (ML) based metamodels (Simpson et al., 2001) may be used instead of the high fidelity models to reduce the overall computing cost.

B. Discrete-Event Simulation.
Discrete processes are characterized by changes in the state of the system occurring only at discrete (countable) instants of time, referred to as *events*. Discrete-event simulation is broadly divided into event-stepped and cycle-stepped approaches. A detailed description

of discrete-event simulation approaches can be found in (Hill, 2007). Formalisms such as Discrete Event System Specification (DEVS) and a subsequent generalization (GDEVS) have been proposed for specification and simulation of discrete-event systems (Zeigler, 1989; Giambiasi et al., 2001). Agalianos et. al. present an overview of issues and challenges for discrete-event simulation in the context of Digital Twins (Agalianos et al., 2020). There exist several proprietary as well as open source libraries and softwares for discrete-event simulations. A review of open source discrete simulation softwares is presented in (Dagkakis and Heavey, 2016).

C. Mixed Discrete-Continuous (MDC) Simulation. Systems containing both discrete-event and continuous processes require a hybrid simulation framework. Kofman proposes a quantization based integration method for simulation of hybrid systems (Kofman, 2004). Nutaro et. al. propose a split system approach in which a-priori knowledge about the discrete-continuous structural split in the model can be used for performing efficient simulation (Nutaro et al., 2012). An approach called Discrete Rate Simulation has been proposed for simulating linear continuous models (such as constant-rate fluid flows) within a discrete-event framework and implemented in a commercial software (Damiron and Nastasi, 2008; Bechard and Cote, 2013). A detailed review of various methodologies used for hybrid discrete-continuous simulation frameworks is presented in (Eldabi et al., 2019). In the following section, we present some design considerations and implementation approaches for MDC simulation, and describe our proposed framework.

3 FRAMEWORK

The system to be modeled can be thought of as a mix of continuous and discrete entities that interact with each other. An *entity* in this context is a collection of state variables, methods and processes representing a particular object to be modeled in the system. A *discrete entity* refers to a process whose state can change only at discrete time instants (events). A *continuous entity* is an entity whose state may be considered to change continuously with time and may require continuous simulation/monitoring. The design of a framework for simulation of such discrete and continuous entities interacting with each other is essentially driven by the following questions:

- How do we advance time?
- How do we simulate interactions between the discrete and continuous entities?

These questions have been addressed by formal approaches proposed for hybrid simulations, for e.g. (Nutaro et al., 2012). From an implementation perspective, the simulation approaches can be broadly classified into two categories as summarized below and illustrated in Figure 1.

(A) In the first approach, the advancement of time is controlled by a single continuous simulation framework. The events to be modeled are embedded into the continuous simulation framework as updates to the state variables or boundary conditions during simulation. These updates may occur at pre-defined time-steps or whenever a certain condition on the state variables is met (for example, when the value of the state variable crosses a particular threshold). The step-size for advancing time is determined by the stability considerations of the numerical scheme used for continuous simulation. If the required step-size differs across multiple continuous entities in the system, the smallest of the step-sizes needs to be used. Such an approach is well suited for systems mainly consisting of tightly coupled continuous entities.

(B) In the second approach, time advancement is handled by an event-stepped discrete-event simulation framework. Continuous solvers are embedded into this framework for simulation of individual continuous entities. Interactions of a continuous entity with other entities in the system are modeled via events. This approach is well-suited when the continuous entities in the system are few and loosely coupled.

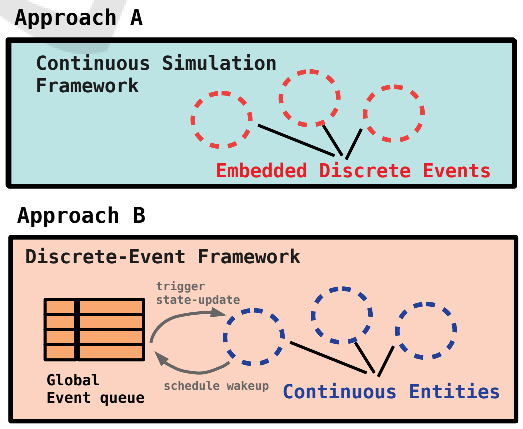
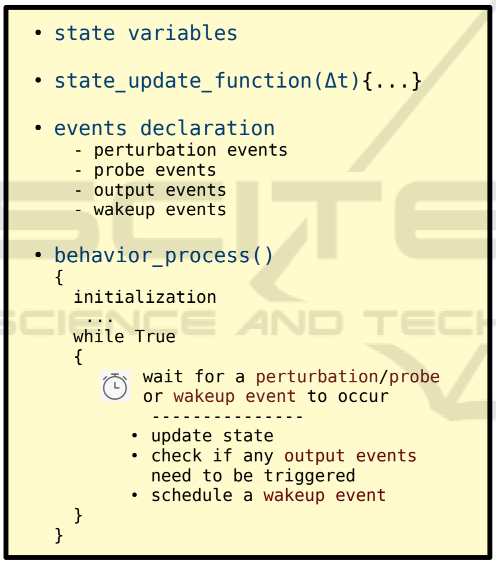


Figure 1: Two approaches to implementation of a MDC simulation framework.

Approach **(B)** is particularly suited for simulation of Digital Twins in manufacturing and process engineering domains since the systems to be modeled are heterogeneous, typically consisting of a larger number of discrete entities and a few continuous entities that are loosely coupled and interact in well-defined ways. We propose a MDC simulation framework based on this approach and describe how the questions of time advancement and capturing interactions between entities are addressed by our framework.

3.1 Proposed Framework

In the proposed framework for MDC simulation, the advancement of time is performed using the event-stepped engine of SimPy. A continuous entity is characterized by its state variables, a state-update function and a definition of events that serve as an interface between the continuous entity and the external world as summarized in Figure 2. The state update function in the continuous entity can serve as a wrapper for performing updates via an external continuous solver.



```
• state variables

• state_update_function(Δt){...}

• events declaration
    - perturbation events
    - probe events
    - output events
    - wakeup events

• behavior_process()
  {
    initialization
     ...
    while True
    {
         wait for a perturbation/probe
         or wakeup event to occur
          ---------------
        • update state
        • check if any output events
          need to be triggered
        • schedule a wakeup event
    }
  }
```

Figure 2: Components of a continuous entity model.

The interactions between a continuous entity and the rest of the system can be implemented via events that serve as the entity's interface. These events are of four types:

1. **Perturbation Event:** An external event that may affect the state/trajectory of the continuous entity.
2. **Probe Event:** An external event which involves querying the state of the continuous entity and thus necessitates updating its state upto a given time.
3. **Output Event:** An event triggered by the continuous entity itself as a consequence of its state update which may affect other entities in the system.
4. **Wakeup Event:** An event scheduled by the continuous entity itself for performing state updates after a fixed time step or for creating output events whose time of occurrence can be predicted in advance.

The behaviour of the continuous entity is modeled as a SimPy process. This process is activated whenever a perturbation, probe or a wakeup event for the entity occurs. When activated,

1. The state updates for the entity upto the current time are computed.
2. If any condition for triggering output events is met, the output events are triggered
3. The continuous entity schedules a wakeup event for itself after a particular time interval. This time interval is determined as follows:
   - (a) If the current trajectory of the state values in the entitity is known, and if all of the output events can be predicted ahead of time based on this trajectory, the wakeup can simply be scheduled at the time the earliest output event is predicted to occur.
   - (b) If all of the output events cannot be predicted ahead of time, the state needs to be updated after fixed number of time steps (possibly on each time step) by scheduling a wakeup event periodically.

At each iteration of the event-stepped algorithm of SimPy, the simulation time is advanced to the timestamp of the earliest scheduled event in the global event list. All events scheduled to occur at this time are executed and the processes waiting on this event are triggered using callbacks.

3.1.1 An Improved Scheme for Time Advancement

One issue with the presented approach is that if the time step size for the continuous simulation is very small relative to the typical time between events in the rest of the system, the cost of adding a wakeup event to the global event list after every time-step can be prohibitive. To address this, we propose a modification as follows:

At each iteration (K) of the event-stepped algorithm, the *tentative* time-step (Δ_K) is taken to be the difference between the scheduled time of the next event in the global event list ($t_{\text{next\_event}}$) and the simulation time for the current iteration(t_K). Each continuous entity is then asked to *peek-ahead* in time by a total period of Δ_K by executing its state-update equations. This can be done either using a single step of size Δ_K or by dividing this period into finer time-steps as as dictated by the time marching scheme. The computation of state updates for multiple continuous entities can potentially be executed in parallel. If no output events of interest are predicted to be generated by any of the continuous entities in this period, the time can be advanced by Δ_K and the computed state-updates in each of the continuous entities can be applied before proceeding to the next iteration. However, if it is found that for a continuous entity i, an event of interest is generated at time $t_i < t_{\text{next\_event}}$, then it may be possible that this event could affect the state or trajectories of the other entities in the system. Thus the actual time-step taken must be the one that advances time to the earliest predicted event across all of the continuous entities. That is, the simulation time should be advanced to $t_{K+1} = \min_i(t_i)$ in the next iteration.

The earliest event predicted to occur at time t_{K+1} can then be inserted into the global event-list, so that its effect on other entities can be propagated as usual in a discrete-event framework, and the state updates in all of the continuous entities computed up to time t_{K+1} can be applied before advancing simulation time to t_{K+1}. A further optimization is to adaptively adjust the tentative time step Δ_K for improved performance.

A proof-of-concept implementation of this framework has been completed and validated using examples. Here, we illustrate the key ideas of the framework using a simple example and present the results generated via simulation.

3.2 An Example

The essential idea of the proposed framework can be illustrated using an example as follows: Suppose the system to be modeled contains multiple fluid tanks whose level is to be simulated and monitored continuously with respect to time. Each tank has an inlet and an outlet valve that can be opened or closed based on external triggers. The flow rate through the inlet/outlet valves can be specified in units of length per time and defined as the maximum change in tank level per unit time when the corresponding valve is open. The flow rates and the maximum level corresponding to the tank capacity can be assumed to be fixed parameters for the tank. The interaction between the tank and other entities in the system can be of two kinds: (a) The opening/closing of the valves can be triggered by external processes and can affect the time-trajectory of the tank level. (b) There are assumed to be external processes in the system that are affected whenever the level of the tank crosses a particular threshold (for example, whenever the tank becomes empty or full). These external processes need to be notified whenever the empty/full conditions occur.

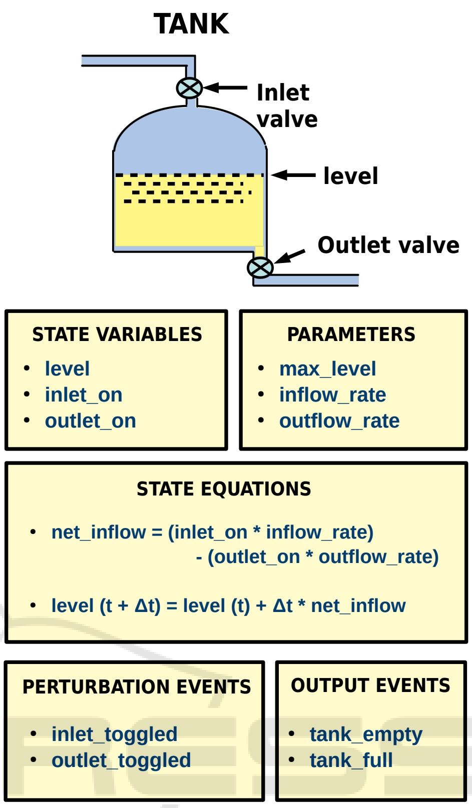


Figure 3: Model of a fluid tank.

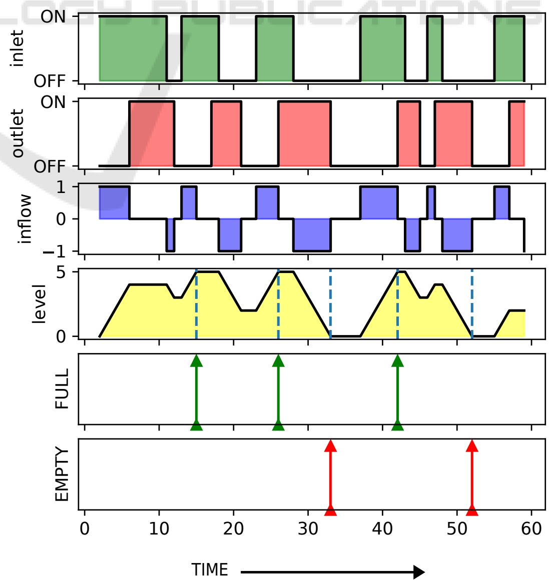


Figure 4: Plots of the time evolution of the tank's state showing the empty/full events generated using the MDC simulation framework.

To model this example in the proposed framework, the tank can be considered a continuous entity and described as a Python class. Objects of this class can be instantiated to create multiple tank instances in the system. The tank entity is characterized by state variables, parameters, and state-update equations as summarized in Figure 3. The opening/closing of valves correspond to perturbation events for the tank and are triggered by an external process. Whenever the tank becomes full/empty a corresponding full/empty output event is triggered by the tank entity. External processes that should be affected by the full/empty events can be automatically notified (by using the `yield <event>` construct of SimPy). The entity also specifies a probe event which can be triggered by external processes. When triggered, it causes the tank state to be updated upto the current time. The behavior of the tank is described as a SimPy process as illustrated in Figure 2. Figure 4 presents the time evolution of the tank's states observed from a simulation run of this example implemented using our framework.

In this particular example, the state update equation is a simple linear algebraic equation and therefore an iterative time marching method is not necessary for continuous state update. Implementing the updates via plain Python code inside the state update method in the tank class suffices and there is no need for invoking an external solver. Further, the exact time instants at which the empty/full events occur can be predicted if the trajectory is not affected by external events. In such a case, the state updates need to be computed only at wakeup events scheduled at time instants when the output events are predicted to occur. Between these time instants, the state trajectory is completely known. In practice, many continuous processes can in fact be replaced by a reduced order surrogate models making the state trajectories known ahead of time. Thus, a simple event-stepped approach can lead to fast and efficient simulation. On the other hand, if the state update equations require iterative time integrators for their solution, it may not be possible to predict the time instants at which output events must occur. Thus, the wakeup events need to be scheduled and the state updates applied periodically (after a fixed number of time-steps).

This example serves to describe the essential components of the proposed framework. The implementation would consist of abstract classes to model continuous entities with an interface provided by the perturbation, probe and wakeup events, and modules for integration with existing continuous solvers.

3.3 Strengths of the Proposed Framework

Some reasons that make this approach particularly suited for application to Digital Twins are as follows:

- The event-stepped approach can result in a more efficient simulation for scenarios where only a few kinds of events affect the trajectory of continuous entities in the system.
- In this approach, it is possible for different continuous entities in the system to use different continuous solvers and internal time step-size values.
- The loose coupling between the continuous entities enables their parallel execution within a single time-step for real-time simulation.
- For modeling entities where a high level of accuracy may not be necessary, coarse surrogate models can be used to predict the trajectory and time of output events and schedule a wakeup ahead of time.
- Sensor value updates from the real system can be easily incorporated into the simulation as perturbation events affecting the state.

4 ROADMAP

We discuss the features of the proposed framework and a roadmap for its implementation along the following aspects:

1. **Integration with Existing Continuous Simulation Frameworks.**
   For fast simulation of continuous processes, integration with established continuous simulation frameworks is necessary. We plan to use Dolfin (Logg and Wells, 2010), a Python based finite-element library for multiphysics modeling and simulation to carry out continuous process simulations within our framework.
2. **Incorporating Analytics.**
   For Digital Twin applications, analytics modules need to be incorporated into the framework for parameter extraction from sensor data, prediction, optimization and for building surrogate models in run-time.
3. **Acceleration for Real-time Simulations.**
   The requirement for real-time simulation creates a need for simulation acceleration that is possible using hardware platforms such as GPGPUs, FPGAs or parallel execution on multi-core systems.

We plan to explore architectures that can take advantage of these technologies for simulations.

4. **Support for Sensing and Control.**
   Sensing and control are integral aspects of a Digital Twin. Features that support these aspects need to be explored in detail and integrated into the framework.

5 CONCLUSIONS

We propose a Python based framework for mixed discrete-continuous simulations specifically targeted for Digital Twins applications. The proposed framework uses SimPy, a Python based discrete-event simulation framework for controlling time-advancement and offers support for integrating existing libraries for continuous process simulation. We present a systematic approach by which the continuous processes can be embedded into the event-stepped discrete simulation engine of SimPy and illustrate the structure of the framework using an example. The ongoing work focuses on further development of the simulation framework on several fronts including, but not limited to: integration with existing continuous solvers, incorporating analytics, real-time simulation and an interface for sensing and control.

REFERENCES

Agalianos, K., Ponis, S. T., Aretoulaki, E., Plakas, G., and Efthymiou, O. (2020). Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics. *Procedia Manufacturing*, 51(2019):1636–1641.

Aversano, G., Ferrarotti, M., and Parente, A. (2020). Digital Twin of a Combustion Furnace Operating in Flameless Conditions: Reduced-Order Model Development from CFD Simulations. *Proceedings of the Combustion Institute*, 000:1–9.

Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., Maier, M., Turcksin, B., and Wells, D. (2016). The `deal.II` Library, Version 8.4. *J. Numer. Math.*, 24:135–141.

Bechard, V. and Cote, N. (2013). Simulation of Mixed Discrete and Continuous Systems: An Iron Ore Terminal Example. In *2013 Winter Simulations Conference (WSC)*, pages 1167–1178.

Cellier, F. E. and Kofman, E. (2006). *Continuous System Simulation*. Springer-Verlag, US.

Chinesta, F., Ladeveze, P., and Cueto, E. (2011). A Short Review on Model Order Reduction Based on Proper Generalized Decomposition. *Archives of computational methods in engineering*, 18:395–404.

Dagkakis, G. and Heavey, C. (2016). A Review of Open Source Discrete Event Simulation Software for Operations Research. *Journal of Simulation*, 10(3):193–206.

Damiron, C. and Nastasi, A. (2008). Discrete Rate Simulation Using Linear Programming. In *Winter Simulation Conference Proceedings*, pages 740–749.

Eldabi, T., Tako, A. A., Bell, D., and Tolk, A. (2019). Tutorial on Means of Hybrid Simulation. *Proceedings of the 2019 Winter Simulation Conference*, pages 273–284.

Feng, L. (2005). Review of Model Order Reduction Methods for Numerical Simulation of Nonlinear Circuits. *Applied Mathematics and Computation*, 167:576–591.

Fritzson, P., Pop, A., Abdelhak, K., Ashgar, A., Bachmann, B., Braun, W., Bouskela, D., Braun, R., Buffoni, L., Casella, F., Castro, R., Franke, R., Fritzson, D., Gebremedhin, M., Heuermann, A., Lie, B., Mengist, A., Mikelsons, L., Moudgalya, K., Ochel, L., Palanisamy, A., Ruge, V., Schamai, W., Sjölund, M., Thiele, B., Tinnerholm, J., and Östlund, P. (2020). The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295.

Fuller, A., Fan, Z., Day, C., and Barlow, C. (2020). Digital Twin: Enabling Technologies, Challenges and Open Research. *IEEE Access*, 8:108952–108971.

Giambiasi, N., Escude, B., and Ghosh, S. (2001). GDEVS: A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems. *Proceedings - 5th International Symposium on Autonomous Decentralized Systems, ISADS 2001*, pages 464–469.

Hill, R. (2007). Discrete-Event Simulation: A First Course. *Journal of Simulation*, 1(2):147–148.

Kofman, E. (2004). Discrete Event Simulation of Hybrid Systems. *SIAM Journal on Scientific Computing*, 25(5):1771–1797.

LeVeque, R. J. (1990). *Numerical Methods for Conservation Laws*. Birkhauser-Verlag, Basel.

Lighthill, M. J. and Whitham, G. B. (1955). On Kinematic Waves II. A Theory of Traffic Flows on Long Crowded Roads. *Proceedings of The Royal Society A*, 229:317–345.

Logg, A. and Wells, G. N. (2010). DOLFIN: Automated Finite Element Computing. *ACM Trans. Math. Softw.*, 37(2).

Molinaro, R., Singh, J. S., Catsoulis, S., Narayanan, C., and Lakehal, D. (2021). Embedding Data Analytics and CFD into the Digital Twin Concept. *Computers and Fluids*, 214:104759.

Nutaro, J., Kuruganti, P. T., Protopopescu, V., and Shankar, M. (2012). The Split System Approach to Managing Time in Simulations of Hybrid Systems Having Continuous and Discrete Event Components. *Simulation*, 88(3):281–298.

Shao, G., Jain, S., Laroque, C., Lee, L. H., Lendermann, P., and Rose, O. (2019). Digital Twin for Smart Manufacturing: The Simulation Aspect. *Proceedings - Winter Simulation Conference*, 2019-Decem(Bolton 2016):2085–2098.

Simpson, T. W., Peplinski, J. D., Koch, P. N., and Allen, J. K. (2001). Metamodels for Computer-based Engineering Design: Survey and Recommendations. *Engineering with Computers*, 17:129–150.

SimPy-Team (2020). Simpy: Discrete-event simulation for python {Online https://simpy.readthedocs.io/en/latest/}.

Sreekumar, M., Joshi, S. M., Chatterjee, A., and Mathew, T. V. (2019). Analyses and Implications of Higher Order Finite Volume Methods on First-Order Macroscopic Traffic Flow Models. *Transportation Letters*, 11:542–557.

Weller, H. G., Tabor, G., Jasak, H., and Fureby, C. (1998). A Tensorial Approach to Computational Continuum Mechanics Using Object-Oriented Techniques. *Computers in Physics*, 12(6):620–631.

Zeigler, B. P. (1989). Devs Representation of Dynamical Systems: Event-Based Intelligent Control. *Proceedings of the IEEE*, 77(1):72–80.