Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

TOWARDS A DEVS-BASED SIMULATION ENGINE FOR DIGITAL TWIN APPLICATIONS

Arnis Lektauers

Institute of Information Technology, Riga Technical University, Riga, LATVIA

ABSTRACT

Digital twins (DT) are increasingly being adopted to improve system monitoring, prediction, and decision making in various domains. Although simulation plays a central role in many DT implementations, a lack of formal modeling foundations often leads to ad hoc and non-scalable solutions. This paper proposes a simulation engine for DT applications based on the Discrete Event System Specification (DEVS) formalism. DEVS provides a robust, modular, and hierarchical modeling framework suitable for modeling the structure and behavior of complex cyber-physical systems. A key contribution is the integration of the Parallel DEVS for Multicomponent Systems (multiPDEVS) formalism with X-Machines to support state and memory separation for simulation models with the goal of improving model scalability and reusability, as well as providing a basis for integration with DTs. The paper presents the architectural design of the engine, highlights its main functional components, and demonstrates its capabilities using a preliminary use case.

1 INTRODUCTION

The integration of information technology into the control of complex cyber-physical systems (CPS) highlights the growing importance of formal simulation-based approaches in the management of complex, data-driven solutions in various sectors, from manufacturing and smart cities to healthcare (Sultanovs et al. 2020), training (Lektauers et al. 2022) and aerospace. The Digital Twin (DT) has emerged as a transformative concept in this context. DT can be defined as a virtual replica of a physical system, continuously synchronized with real-world data, allowing simulation and analysis of the system's behavior in parallel with its operation (Iliuţă et al. 2024).

By coupling sensor data with models, DTs allow one to monitor current conditions in real time and predict future states under various scenarios. In practice, this means that a DT can run simulations to forecast performance, anticipate faults or maintenance needs, and evaluate 'what if' scenarios, thus supporting proactive decision-making and optimization in complex systems. DTs are gaining traction across industries, reflecting their potential to improve the operational efficiency of complex systems, reduce unexpected breakdowns, and support smarter decision making based on real data simulations. As a result, simulation plays a critical role in DT development, serving as one of its key enabling technologies (Biller et al. 2022).

However, with current DT simulation practices, achieving the benefits mentioned above is difficult. Many existing DT implementations are developed in an ad hoc manner, without a unified or formalized approach. This can lead to inconsistencies and an increased risk of errors during the development process (Niyonkuru and Wainer 2021). The lack of a consistent and unified understanding of what constitutes a DT (Traoré 2021), combined with the lack of an appropriate formalism, complicates the verification and testing of DTs. As a result, it is also difficult to ensure that the virtual representation remains consistent with its physical counterpart over time. These problems become exponentially harder when scaling up, for example, when DT concepts are extended to the scale of smart factories or cities. In such cases, interoperability and scalability issues arise. Many simulation components are being built in isolation, making them difficult to reuse or combine with other systems and limiting their modularity (Rasheed et al. 2020). Another challenge

is achieving real-time performance. Integrating high-fidelity models with real-time data streams can place stringent demands on computational resources and network bandwidth, which affects the ability to realize synchronized and responsive simulations. What is needed is a more systematic approach to building DTs, where formal semantics, modular design, and efficiency are built in from the start.

Although no single simulation methodology can satisfy all DT application requirements, several paradigms align well with DT-specific needs, including real-time responsiveness, multiscale modeling, and integration with live data. Table 1 provides an overview of the simulation methods commonly applied in DT contexts, reflecting a wide spectrum of approaches, from discrete event modeling to hybrid and agent-based simulations (Yao et al. 2023; Vanommeslaeghe et al. 2024; Wooley et al. 2023). Among the various approaches for the simulation modeling of DTs cited in scientific sources, Agent-Based Modeling and Simulation (ABMS) stands out for its integration with Multi-Agent Systems (MAS) that have many similarities with DT (Pretel et al. 2024). The flexibility of ABMS to capture adaptive behavior complements the robust time management and integration capabilities of DEVS.

Table 1: Simulation modeling methods commonly used in DT applications.

Simulation Method	**Main Capabilities**	**Typical Application Domains**
Discrete Event Simulation (DES)	Event-driven approach	Manufacturing, logistics (Biller et al. 2022)
Discrete Event System Specification (DEVS)	Modular simulation architecture; event-driven, real-time hybrid execution support	Manufacturing, logistics, healthcare, CPS (Niyonkuru and Wainer 2021; Zeigler et al. 2018)
Cellular Automata (CA)	Emergent behavior analysis	Emergency response systems, urban planing (Foures et al. 2018)
Agent-Based Modeling & Simulation (ABMS)	Emergent behavior; decentralized control; adaptive scenarios	Transportation, smart grid, emergency response, urban planning (Ambra and Macharis 2020; Pretel et al. 2024)
System Dynamics	Feedback-driven modeling; multidisciplinary integration; low computational complexity	Defense, maintenance planning, policy analysis, lifecycle simulation (Choi et al. 2023)
Continuous Simulation (ODE/PDE)	Physics-based modeling; real-time control	Aerospace, automotive, energy systems (Tian et al. 2023)
Co-Simulation	System integration; cross-domain fidelity	Smart manufacturing, CPS, robotics (Vanommeslaeghe et al. 2024)
Hybrid Simulation	Synchronization of different time models; holistic representation of processes	Autonomous systems, process control, manufacturing (Farsi et al. 2019)
Data-Driven Simulation / Surrogate Modeling	Fast estimation; model calibration; predictive analytics	Prognostics, digital health, structural monitoring (Chakraborty et al. 2021; Tian et al. 2023; Yao et al. 2023)
Real-Time Simulation / Hardware-in-the-Loop (HIL)	Synchronization with physical systems and control loops	Automotive, aerospace, robotics (Jiménez Aparicio and Roßmann 2025)

This paper proposes a DEVS-based simulation engine to support digital twin applications. Based on the DEVS formalism, the proposed engine aims to overcome the shortcomings of current DT simulation practices: it provides a formal, modular approach to modeling complex twins; it ensures scalability through

hierarchical decomposition and efficient event-based execution; and it enables the seamless integration of discrete event simulation and ABMS. The following sections discuss in detail the concept and architecture of the DEVS-based DT simulation engine, including formal definitions of its modeling components, the design of the service-oriented platform, the implementation of multi-agent extensions, and preliminary results from a case study.

2 DEVS-BASED SIMULATION

The Discrete Event System Specification (DEVS), introduced by Zeigler in 1976 (Zeigler 1976), is a widely recognized formalism for modeling and simulating discrete-event systems. DEVS offers a mathematically rigorous foundation that enables systems to be modeled as modular and hierarchically composed components, each defined by their state transition logic and event scheduling behavior (Zeigler 1976; Zeigler et al. 2018). Based on a generic and extensible structure, the DEVS formalism can be adapted to a wide range of simulation paradigms, including differential equations, cellular automata, and hybrid models, making it suitable for the simulation of CPS (Wainer et al. 2018), combining discrete and continuous dynamics.

Several studies have explored the role of DEVS simulation in DT solutions. For instance, Niyonkuru and Wainer (2021) proposed a DEVS-based architecture for digital quadruplets, emphasizing modularity and the ability to integrate real-time data into simulation-based components. Their work establishes the feasibility of using DEVS for live synchronized virtual-physical models, although it does not support agent-level modeling. Similarly, Vanommeslaeghe et al. (2024) demonstrated how DEVS can be co-simulated with embedded systems using the Functional Mock-up Interface (FMI) 3.0, suggesting an increasing interest in using DEVS for operational DT scenarios. In a complementary direction, David et al. (2021) introduced a conceptual framework to infer DEVS models using reinforcement learning, with the aim of automating and generalizing the construction of simulation models in different DT contexts.

In DEVS, a complex system is decomposed into submodels (atomic and coupled models) that interact via discrete events, which naturally facilitates modularity and reusability. Atomic models encapsulate behavior through internal, external, and confluent transition functions along with an output function and a time advance function. Coupled models, in contrast, define the structure and interconnections of submodels without directly encoding behavior. This separation of behavior and structure facilitates composability and reuse. However, in classic DEVS, coupled models are static containers that cannot represent their own internal state or behavior directly. This limitation has motivated various extensions, such as ML-DEVS (Steiniger and Uhrmacher 2016) and EB-DEVS (Foguelman, Henning, Uhrmacher, and Castro 2021), to support dynamic model structures and emergent behavior.

2.1 multiPDEVS Approach for Multicomponent Simulation

Although classic DEVS provides solid foundations for modeling modular systems, it is limited in expressing dynamic structures or high-volume interactions, particularly in multi-component settings. To address these challenges, the author adopts the multiPDEVS (Parallel DEVS for Multicomponent Systems) (Foures et al. 2018) formalism.

multiPDEVS is an extension of the classic DEVS, Parallel DEVS (PDEVS) and Multicomponent DEVS (multiDEVS) formalisms (Zeigler et al. 2018), designed to support parallel discrete event processing across multicomponent systems, nonmodular decomposition, and efficient handling of simulation event collisions and conflicts. According to Foures et al. (2018), a multiPDEVS model is defined as a set of interacting components $\{M_d\}$:

$$multiPDEVS = (X, Y, D, \{M_d\}), \tag{1}$$

where X and Y are sets of input and output events, D is the set of component references.

For each $d \in D$, a component M_d is defined as a tuple:

$$M_d = (S_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \delta_{con,d}, \delta_{reac,d}, \lambda_d, ta_d), \tag{2}$$

where S_d is the set of sequential states of d, $I_d \subseteq D$ is the set of influencing components, and $E_d \subseteq D$ is the set of influenced components, $Q_i = \{(s,e_d)|s \in S_d, e_d \in \mathbb{R}_0^+, \quad 0 \leq e_d \leq ta_d(s_d)\}$ is the set of total states with e the time elapsed since the last transition.

$ta_d : \underset{i \in I_d}{\times} Q_i \to \mathbb{R}_0^+ \cup \{\infty\}$ is the time advance function to schedule the time of the next internal event based on total states of the influencers.

$\lambda_d : \underset{i \in I_d}{\times} Q_i \to Y^b$ is the output function, where Y^b is the set of output bags.

$\delta_{int,d} : \underset{i \in I_d}{\times} Q_i \to \underset{j \in E_d}{\times} S'_j$ is the internal transition function that maps the total state of the influencers into suggested states S' for the set of influencees.

$\delta_{ext,d} : \underset{i \in I_d}{\times} Q_i \times X^b \to \underset{j \in E_d}{\times} S'_j$ is the external transition function, where X^b is the set of input bags.

$\delta_{reac,d} : K_d^b \times Q_d \to S_d$ is the reaction transition function, where K_d^b is the set of bags of suggested states for d over elements in K_d. $K_d = \{(s_d,c)|s_d \in S'_d, d \in E_c\}$, where the tuple (s_d,c) is a suggested state for $d \in E_c$ produced by the component c.

$\delta_{con,d} : \underset{i \in I_d}{\times} Q_i \times X^b \to \underset{j \in E_d}{\times} S'_j$ is the confluent transition function (originally introduced in PDEVS).

2.1.1 Component Memory Extension

A common challenge in multiPDEVS is the requirement for a component to recompute the entire state of an influencee even to change a single state variable. To address this, the author incorporates an explicit memory structure $\mathbb{M}$ as a formal extension to the components states S_d. A similar approach, for example, is used in the formalism X-Machines (Extended Finite-State Machines with Memory and Functions) (Kefalas et al. 2003).

Formally, DEVS and X-Machines represent two complementary formal modeling approaches. X-Machines extend classical finite state machines by incorporating an internal memory structure $\mathbb{M}$. The difference between states and memory sets allows flexibility in modeling systems because there are situations where models have only one or few internal states and multiple complex variables stored in memory (Kiran 2017).

In that context, the original set of sequential states S_d of the component M_d (2) the author proposes to replace by a tuple of a finite set of states $\mathbb{S}_d$ and (possibly) large set of memory values $\mathbb{M}_d$: $S_d = (\mathbb{S}_d, \mathbb{M}_d)$. The revised multiPDEVS component definition becomes the following:

$$M_d = (\mathbb{S}_d, \mathbb{M}_d, I_d, E_d, \delta_{ext,d}, \delta_{int,d}, \delta_{con,d}, \delta_{reac,d}, \lambda_d, ta_d). \tag{3}$$

Separation of state logic and memory allows for more compact state representations, more precise control of simulation transitions, and reduced overhead during state propagation. It also provides a natural mechanism to implement agent models with contextual awareness and decision memory, aligning well with ABM and DT applications that require adaptive behavior and where agents / components have few control states (e.g., Idle, Active) and rich internal memory (e.g., battery level, path history, local temperature, etc.).

In Figure 1, a simple abstract example is shown that defines a smart light control agent that implements the revised multiPDEVS component (3). The agent operates in one of three control states: IDLE (no presence detected), ACTIVE (person detected), and ALERT (motion detected but light malfunctioning). It maintains internal memory, including *occupancy* (representing the motion detection), *lightStatus*, and *lastSwitchTime*. A transition function governs the agent's behavior based on current memory values and conditions such as elapsed time. The agent performs actions such as turning the light on or off or sending an alert and transitions between states accordingly. The diagram illustrates the separation between control logic (states) and data context (memory), showing how decisions emerge from their interaction via state transitions.

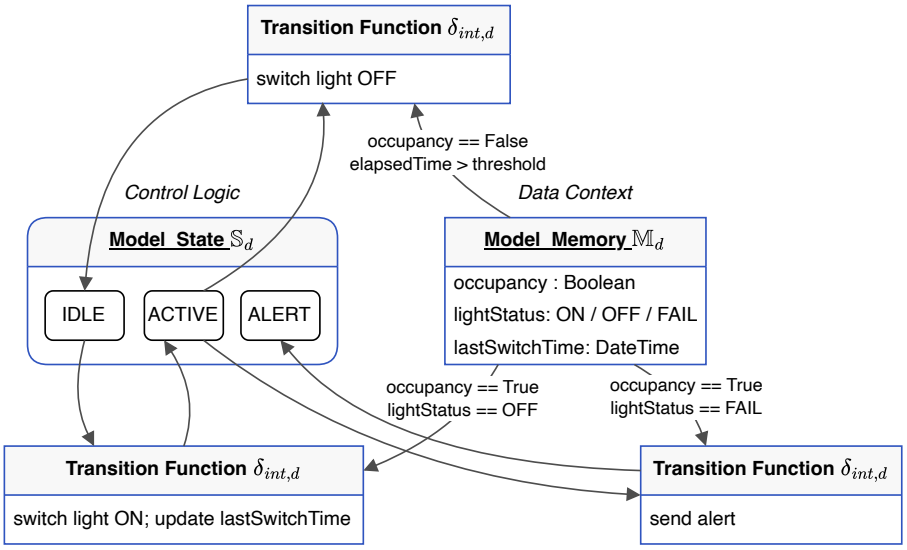


Figure 1: Abstract model example of the separation between control logic (states) and data context (memory).

3 MODELING AND SIMULATION AS A SERVICE

To support the full lifecycle of digital twin (DT) applications, the proposed simulation platform is designed following the Modeling and Simulation as a Service (MSaaS) paradigm (Procházka and Hodický 2017). This architecture facilitates modular, distributed, and interoperable simulation components accessible via service-based interfaces.

Figure 2 illustrates the conceptual operational architecture of the platform, depicting a closed-loop DT value chain (Traoré 2021). At the core of this architecture lies the modeling and simulation layer, which exposes simulation functionality via APIs using standard web protocols such as HTTP and WebSockets. This structure enables seamless integration with external systems and user interfaces, fostering both machine-to-machine communication and human-in-the-loop interaction.

3.1 Simulation Engine Architecture

The DEVS-based simulation engine (Figure 3) is designed to operate within the MSaaS environment and serves as the computational backbone of the DT platform. The architecture follows a modular and service-oriented structure designed to support DT development and operation in distributed environments. At its core, the simulation engine implements the two main parts: the formal model and the model simulator. These are coupled via the modeling and simulation relations, ensuring a clear separation of concerns between representation and execution. In general, this architecture provides a flexible and interoperable simulation framework to implement complex DT scenarios.

The key component of the engine is the Simulator that integrates a high-performance event scheduler, enabling scalability for large-scale, real-time DT applications. The simulation back-end supports multi-PDEVS and multi-agent simulation constructs, facilitating fine-grained concurrency and dynamic model structures. To support experimentation, the engine incorporates the Experimental Frame (EF), which comprises a Generator to produce input data, an Acceptor to filter the simulation scenario, and a Transducer to summarize the results (Zeigler et al. 2018).

In terms of model management, the engine supports models defined in a Simulation Model Definition Language (SMDL), inspired by the Digital Twins Definition Language (DTDL) (Microsoft Azure 2025), which is in work in progress by the engine authors and allows seamless integration with IoT and DT

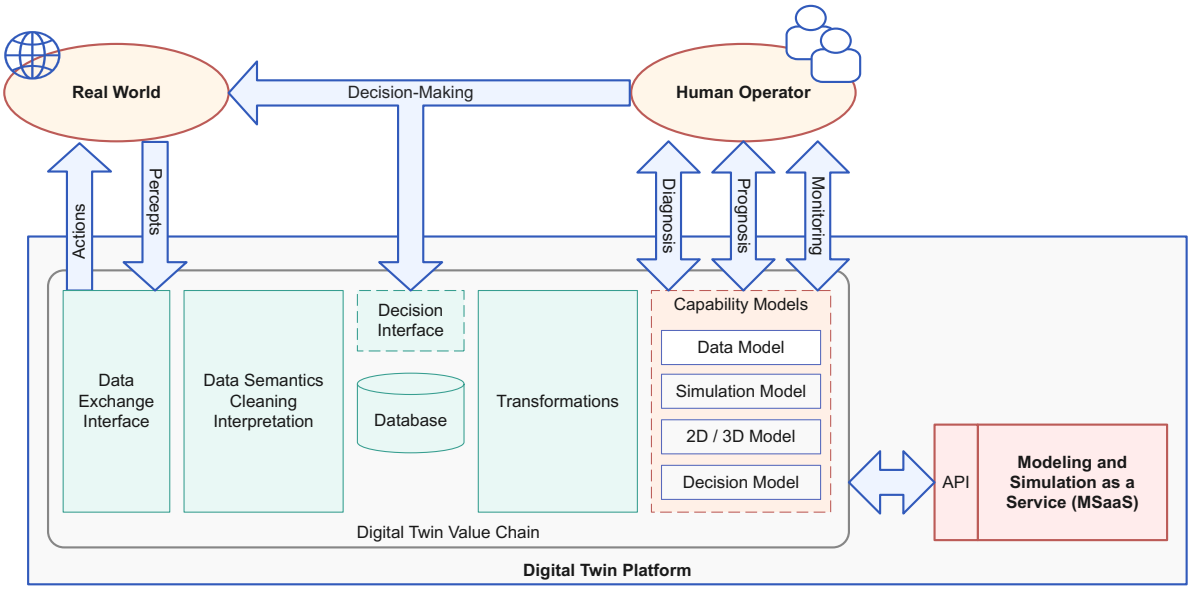


Figure 2: Generic DT platform concept based on operational architecture principles for a DT value chain (adapted from Traoré 2021).

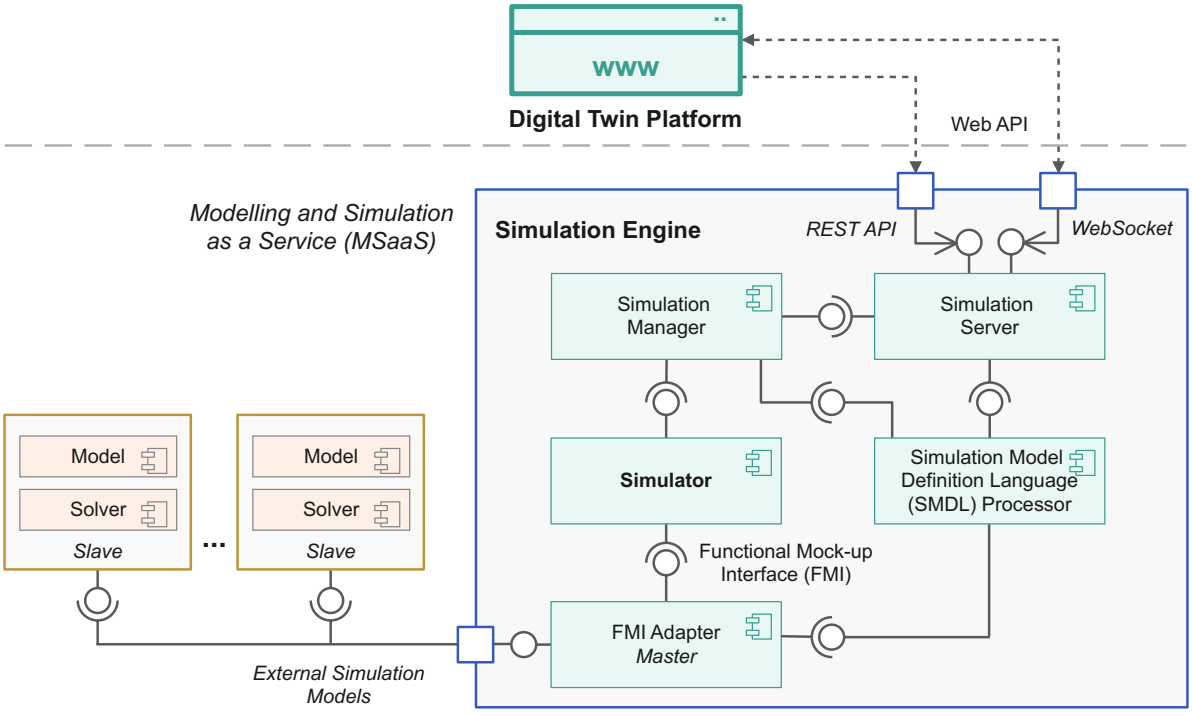


Figure 3: Architecture of the simulation engine implementing the MSaaS paradigm.

infrastructures. The SMDL Processor handles model specification input – it parses a model description to construct the DEVS models in the engine. Using an SMDL, users can define their twin simulation model, which the processor then translates into instantiated model elements.

The Simulation Manager manages and coordinates the execution of the simulation, ensuring correct timing and coordination of interactions. Its main task is to ensure the scalability of the simulation by using the available heterogeneous computing (CPU / GPU) resources as needed. The manager communicates with the FMI adapter (to step external models) and relies on the SMDL processor for the initial model setup.

Users can define models using DEVS or provide external Functional Mock-up Units (FMUs) supported via a built-in FMI adapter. This hybrid capability allows for co-simulation with third-party tools and supports a wide range of modeling paradigms including discrete-event, continuous, and hybrid systems. The FMI adapter is responsible for co-simulation, enabling the engine to load FMUs and orchestrate their execution alongside DEVS models, conceptually similar, as described in Vanommeslaeghe et al. (2024).

3.2 Real-Time Data Loop Support and Collaborative Simulation

DTs often operate as part of a cyber-physical feedback loop, where simulations must ingest real-time data from physical systems and return actionable outputs. The proposed engine supports this loop through the MSaaS architecture:

- Input handling: Real-time input streams are supported via RESTful API and WebSocket endpoints, enabling external systems to push sensor data or events into the simulation model at run-time.
- Output generation: Simulation responses, for example, predicted failures, control signals, simulation statistics, or agent decisions, are exposed through the WebSocket endpoint to external subscribers.
- Clock synchronization: DEVS's explicit handling of simulation time allows the engine to be synchronized with wall-clock time when necessary, enabling accurate simulation aligned with physical system operations.

Since each component of the model maintains its own timing logic, incoming updates can be scheduled with precise control over how and when they impact the simulation. This makes the approach suitable for industrial DT deployments where data frequency, causality, and timing guarantees are critical (e.g., smart grids, production cells, or autonomous systems).

The simulation engine can be used to run multiple collaboratively interacting DTs, each using a separate simulation engine instance. The interacting DTs can communicate by exchanging events, involving an external simulation orchestrator, using RESTful APIs and WebSocket endpoints on each simulation engine instance. This approach enables coordinated behavior, such as cooperative decision-making or shared resource management, as well as supports scalability and facilitates the creation of interconnected DT ecosystems consisting of interacting fragments suitable for system-of-system modeling.

4 IMPLEMENTATION AND PRELIMINARY TESTING

Figure 4 presents a UML class diagram of the implementation of the simulation engine, highlighting the key classes and their relationships in the multiPDEVS-based and agent-based design. The open source implementation in the Kotlin language is divided into five main packages: `Common` and `Simulation` packages defining generic simulation abstractions, DEVS and multiPDEVS packages implementing the extended discrete event formalism, and an Agents package supporting multi-agent modeling.

At the core, an `AbstractModel` mixin implements `Simulable`, `Couplable`, `Observable` and `Stateful` interfaces. A conceptually similar mixin-based approach is used in the Quartz (A Crystal Modeling & Simulation Framework) discrete event simulation library (Franceschini et al. 2018). `AbstractModel` is a container for model variables (implemented by the `Variable` class) and states (implemented by the `State` class).

The `Component` class in the `multiPDEVS` package implements a multiPDEVS component formally defined by (1). The `Model` class in the same package implements a multiPDEVS model formally defined by (3).

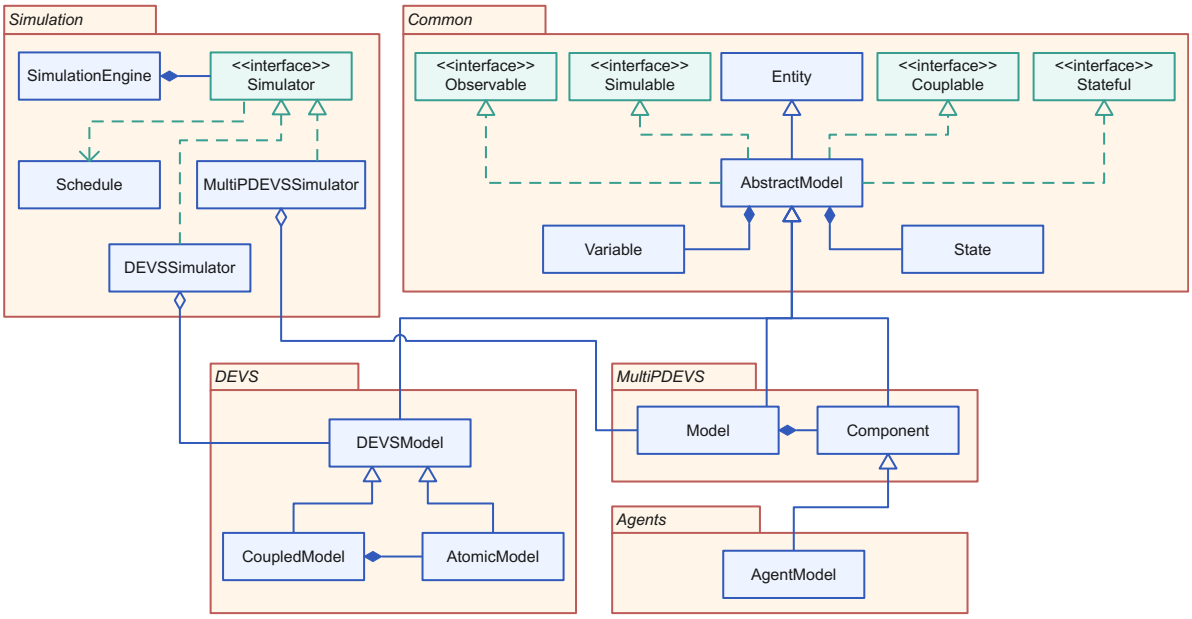


Figure 4: Simulation engine packages and core classes.

The `SimulationEngine` class orchestrates the overall execution, coordinating `Simulator` objects (each implementing the `Simulator` interface). A central `Schedule` structure maintains the future event list, using a binary heap (Nutaro 2011).

4.1 Multi-Level Multi-Agent Simulation

The engine implementation strategy enables multilevel multi-agent simulation by bridging the gap between ABMS and the system-theoretic DEVS approach. In contrast to earlier hybrid or transformation-based simulation frameworks for complex systems (Seddari et al. 2021; Farsi et al. 2019), the implemented approach integrates agents directly into the discrete event simulation core, allowing DT models to incorporate both strict system dynamics and flexible agent behaviors in one platform. The design of the agent-based simulation package is influenced by FLAME GPU which is a GPU-accelerated simulator for domain-independent complex system simulations (Kiran 2017; Richmond et al. 2023).

The package `Agents` (Figure 5) introduces `AgentModel`, `Population`, `Agent`, `Environment`, `Layer` and `Function` classes to enable multi-agent and multilevel simulation within the DEVS framework.

Each `AgentModel` represents a specialized multi-PDEVS component that acts as a simulator of agent populations (`Population` class) using a time-stepped approach at a defined time resolution level. Each `Population` contains many agents (`Agent` class) of the same type. Each agent interacts within a given `Environment`. The coupled set (`AgentModel`, `Agent`, `Environment`, `Population`) partially corresponds to the Influence Reaction Model for Multi-Level Simulation (IRM4MLS) (Morvan and Jolly 2012). In comparison to IRM4MLS, in this implementation, each agent belongs to only one simulation level defined by the `AgentModel`.

The X-Machines formalism is adopted for `Agent` design, which extends finite-state machines with an internal memory component $\mathbb{M}$ separating a finite control state $\mathbb{S}$ and a (potentially large) memory $\mathbb{M}$ for variables and attributes, as described in Subsection 2.1.1. Each step of `Agent` instance execution is implemented by one or more sequential functions (`Function` class). The functions are automatically organized in layers (`Layer` class) based on their dependencies, which allows functions in individual layers to potentially be executed in parallel (e.g., on a GPU).

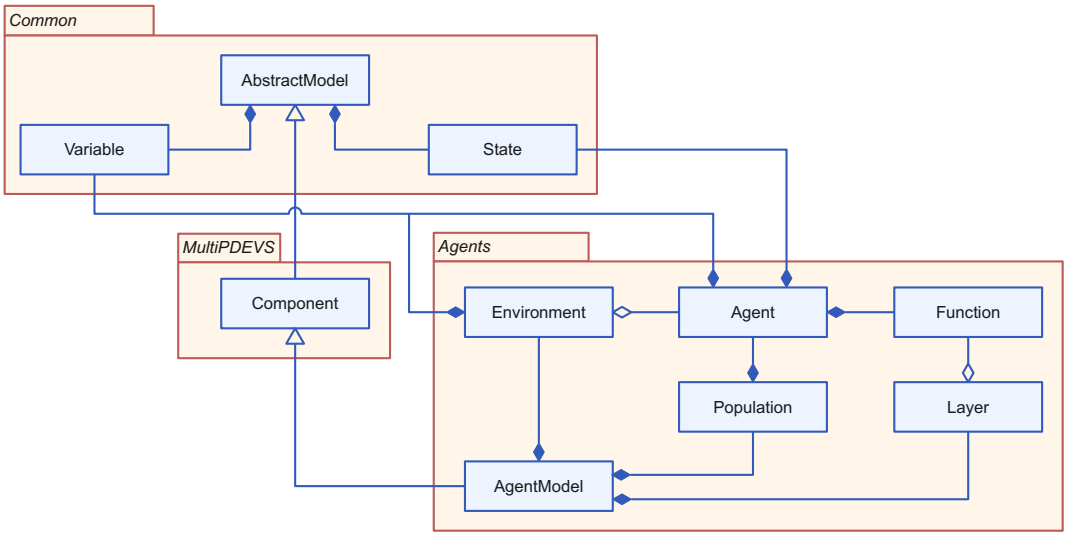


Figure 5: Diagram of ABMS classes.

4.2 Testing: Case Study of Fire Spread Modeling

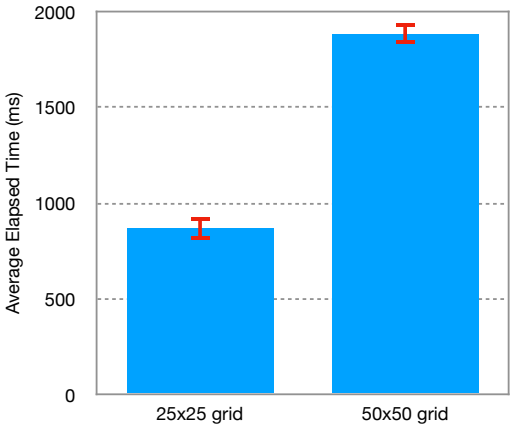
To evaluate the implemented simulation engine, a preliminary case study was conducted involving a simple scenario of fire spread modeling, which is described by the authors of multiPDEVS (Foures et al. 2018) and is also implemented in the Quartz discrete event simulation library (Franceschini et al. 2018).

The fire spread case study models a 2D grid-based environment where each grid cell represents a forest unit (or patch) and can be in one of three states: *unburned*, *burning*, or *burned*. In each simulation step, the fire spreads from burning cells to neighboring unburned cells with a predefined probability. The transition dynamics follows discrete event timing; each cell has an internal time variable governing how long it remains in the burning state before transitioning to burned. This simple model captures emergent spatial dynamics, such as the propagation of the fire front. The model is usable for benchmarking purposes due to its controllable complexity, modular cell behavior, and high interaction density between components. In the implementation, each cell is modeled as an agent with state-memory separation, allowing context-aware spreading and timing behavior.

Simulation runs were carried out on a test environment consisting of an Apple M2 Max CPU, 32 GB of RAM, running on OSX 15.4.1. Two grid sizes were tested to examine scalability: a 25×25 cell grid (625 agents) and a 50×50 cell grid (2500 agents). In each case, the simulation was executed for a fixed virtual duration of 1200 time units (sufficient for the fire to spread and burn out) and repeated for 10 independent runs. In this experiment, the scheduling of the engine was single-threaded (using one logical `SimulationEngine` instance managing all events sequentially).

Figure 6 presents a bar graph of the average execution times, showing that the 50×50 case takes approximately five times longer than the 25×25 case. This scaling is in line with expectations: the larger grid has 4x the number of agents and significantly more interactions, leading to a super-linear increase in processing time due to the overhead of managing more events. However, even for 2500 agents and a 1200 steps of simulation, the total execution time remained under 2 seconds, suggesting that the simulation engine is suitable for complex real-time or near-real-time applications. These preliminary performance tests demonstrate that the DEVS-based engine can efficiently handle components or agents of the order 10^3-10^4 and their interactions in real time or near real time, on a single modern laptop-class processor. The engine maintained correct time synchronization and event ordering throughout the runs, and the fire spreading dynamics was visually verified to match the expected behavior.

These preliminary performance tests demonstrate the engine's ability to efficiently manage thousands of active components and their interactions. A GPU-accelerated scheduler was not directly needed at this



Grid	Average Elapsed Time (ms)	Relative Standard Deviation (%)
25x25 grid	869,14	4,65
50x50 grid	1885,899	2,93

Figure 6: Average wall-clock execution time (milliseconds) for the fire spread simulation on grids of size 25×25 and 50×50 (1200 time steps, 10 runs per scenario). Error bars indicate relative standard deviation across runs.

scale, but the infrastructure is designed for larger scenarios. It is anticipated that more complex agents (with more functions or layers) or larger scale models (e.g., 100k agents) will require the use of GPUs and parallel execution.

5 CONCLUSION

This paper has presented a formal, modular simulation engine architecture grounded in the DEVS formalism, designed to meet the growing demands of DT applications. Using a system theoretical framework and based on the MSaaS paradigm, the engine supports scalable and time-consistent integration of DT components. The integration of an FMI adapter enables interoperability with diverse modeling ecosystems.

A key contribution of this work is the integration of multiPDEVS with the X-Machines formalism that enables the modeling of MAS with memory, context awareness, and decision-making logic within a formally verifiable and composable simulation environment. As a result, the engine supports both well-structured system decomposition and flexible agent-based behaviors - capabilities essential for representing modern CPS and DTs. The developed simulation engine provides a solid foundation for DTs that require hybrid modeling, multilevel abstraction, and dynamic scenario evaluation.

Although the tested fire spread model provides a controlled environment to test the core functionality and scalability of the engine, it does not fully reflect the complexity or data-driven dynamics of industrial DT applications. The fire spread use case serves primarily as a proof-of-concept to demonstrate functional correctness, component integration, and basic performance scaling of the proposed engine. Therefore, as part of the ongoing work, the author is preparing validations using more representative DT scenarios from the manufacturing and transportation domains. In particular, the plan is to apply the engine to more complex case studies (e.g., a public transport DT) to evaluate its performance and scalability in realistic settings.

Future versions of the engine will be benchmarked against classical DEVS engines (e.g., Quartz), EB-DEVS, and ABMS-focused frameworks such as FLAME GPU based on the hypothesis that the integration of X-Machines can provide better scalability of large models due to reduced state transition overhead and improved separation of logic between states and memory.

Significant attention in the future will also be paid to the use of GPUs for simulation acceleration, as well as to improving the integration of ABMS with multiPDEVS. Extending the engine with adaptive machine learning components, such as those that can update agent behavior or model parameters at runtime, allowing models to be adjusted online as new data is received, is also being explored.

By connecting formal modeling with practical deployment needs, the proposed engine paves the way for more reliable, reusable, and intelligent digital twin simulations.

ACKNOWLEDGEMENTS

This research is conducted as part of the project "Development of the DigiTDevOps digital twin development and operation platform" under the European Union's Recovery and Resilience Mechanism Plan (Project Number: 5.1.1.2.i.0/4/24/A/CFLA/001).

REFERENCES

Ambra, T., and C. Macharis. 2020. "Agent-Based Digital Twins (ABM-DT) in Synchromodal Transport and Logistics: The Fusion of Virtual and Physical Spaces". In *2020 Winter Simulation Conference (WSC)*, 159–169 https://doi.org/10.1109/WSC48552.2020.9383955.

Biller, B., X. Jiang, J. Yi, P. Venditti, and S. Biller. 2022. "Simulation: The Critical Technology in Digital Twin Development". In *2022 Winter Simulation Conference (WSC)*, 1340–1355 https://doi.org/10.1109/WSC57314.2022.10015246.

Chakraborty, S., S. Adhikari, and R. Ganguli. 2021. "The Role of Surrogate Models in the Development of Digital Twins of Dynamic Systems". *Applied Mathematical Modelling* 90:662–681.

Choi, J., S. Moon, and S. Min. 2023. "Digital Twin Simulation Modeling Process with System Dynamics: An Application to Naval Ship Operation". *International Journal of Robust and Nonlinear Control* 33(16):10136–10150.

David, I., J. Galasso, and E. Syriani. 2021. "Inference of Simulation Models in Digital Twins by Reinforcement Learning". In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. October 10–15, Fukuoka, Japan, 221–224.

Farsi, M., J. Erkoyuncu, D. Steenstra, and R. Roy. 2019, February. "A Modular Hybrid Simulation Framework for Complex Manufacturing System Design". *Simulation Modelling Practice and Theory* 94:14–30.

Foguelman, D., P. Henning, A. Uhrmacher, and R. Castro. 2021, May. "EB-DEVS: A Formal Framework for Modeling and Simulation of Emergent Behavior in Dynamic Complex Systems". *Journal of Computational Science* 53:101387.

Foures, D., R. Franceschini, P.-A. Bisgambiglia, and B. P. Zeigler. 2018, January. "multiPDEVS: A Parallel Multicomponent System Specification Formalism". *Complexity* 2018(1):1–19.

Franceschini, R., P.-A. Bisgambiglia, P. Bisgambiglia, and D. R. C. Hill. 2018. "An Overview of the Quartz Modelling and Simulation Framework". In *Proceedings of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, Volume 1, 120–127. Setubal, Portugal: SCITEPRESS - Science and Technology Publications, Lda.

Iliuţă, M.-E., M.-A. Moisescu, E. Pop, A.-D. Ionita, S.-I. Caramihai, and T.-C. Mitulescu. 2024. "Digital Twin—A Review of the Evolution from Concept to Technology and Its Analytical Perspectives on Applications in Various Fields". *Applied Sciences* 14(13):5454.

Jiménez Aparicio, J. L., and J. Roßmann. 2025. *Experimentable Digital Twins in the Loop*, 195–206. Cham: Springer Nature Switzerland.

Kefalas, P., G. Eleftherakis, and E. Kehris. 2003. "Communicating X-Machines: From Theory to Practice". In *Advances in Informatics*, edited by Y. Manolopoulos, S. Evripidou, and A. C. Kakas, 316–335. Berlin, Heidelberg: Springer Berlin Heidelberg.

Kiran, M. 2017. *X-Machines for Agent-Based Modeling: FLAME Perspectives*. Boca Raton: Chapman and Hall/CRC.

Lektauers, A., J. Bikovska, and V. Bolsakovs. 2022. "An Agent-Directed Digital Twin Framework for Simulation-Based Training". In *2022 63rd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*, 1–6. Riga: Institute of Electrical and Electronics Engineers (IEEE).

Microsoft Azure 2025. "Digital Twins Definition Language (DTDL)". https://github.com/Azure/opendigitaltwins-dtdl. Accessed: April 17, 2025.

Morvan, G., and D. Jolly. 2012. "Multi-Level Agent-Based Modeling with the Influence Reaction Principle". *CoRR* abs/1204.0634.

Niyonkuru, D., and G. Wainer. 2021, April. "A DEVS-Based Engine for Building Digital Quadruplets". *SIMULATION* 97(7):485–506.

Nutaro, J. 2011. *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, New Jersey: Jon Wiley & Sons.

Pretel, E., A. Moya, E. Navarro, V. López-Jaquero, and P. González. 2024. "Analysing the Synergies Between Multi-Agent Systems and Digital Twins: A Systematic Literature Review". *Information and Software Technology* 174:107503.

Procházka, D., and J. Hodický. 2017. "Modelling and Simulation as a Service and Concept Development and Experimentation". In *2017 International Conference on Military Technologies (ICMT)*, 721–727. Brno: Institute of Electrical and Electronics Engineers (IEEE).

Rasheed, A., O. San, and T. Kvamsdal. 2020. "Digital Twin: Values, Challenges and Enablers from a Modeling Perspective". *IEEE Access* 8:21980–22012.

Richmond, P., R. Chisholm, P. Heywood, M. K. Chimeh, and M. Leach. 2023. "FLAME GPU 2: A Framework for Flexible and Performant Agent Based Simulation on GPUs". *Software: Practice and Experience* 53(8):1659–1680.

Seddari, N., S. Boukelkoul, A. Bouras, M. Belaoued, and R. Mohamed. 2021. "A New Transformation Approach for Complex Systems Modelling and Simulation: Application to Industrial Control System". *International Journal of Simulation and Process Modelling* 16(1):34–48.

Steiniger, A., and A. M. Uhrmacher. 2016, January. "Intensional Couplings in Variable-Structure Models: An Exploration Based on Multilevel-DEVS". *ACM Transactions on Modeling and Computer Simulation* 26(2):1–27.

Sultanovs, E., J. Strebko, A. Romanovs, and A. Lektauers. 2020. "The Information Technologies in the Control Mechanism of Medical Processes". In *2020 61st International Scientific Conference on Information Technology and Management Science of Riga Technical University*, 1–5. Riga: Institute of Electrical and Electronics Engineers (IEEE).

Tian, H., H. Zhao, H. Li, X. Huang, X. Qian, and X. Huang. 2023. "Digital Twins of Multiple Energy Networks Based on Real-Time Simulation using Holomorphic Embedding Method, Part II: Data-Driven Simulation". *International Journal of Electrical Power & Energy Systems* 153:109325.

Traoré, M. K. 2021. "Unifying Digital Twin Framework: Simulation-Based Proof-of-Concept". *IFAC-PapersOnLine* 54(1):886–893.

Vanommeslaeghe, Y., B. Van Acker, J. Denil, and P. De Meulenaere. 2024. "Integrating DEVS and FMI 3.0 for the Simulated Deployment of Embedded Applications". In *2024 Annual Modeling and Simulation Conference (ANNSIM)*, edited by J. O. Bentley and R. C. Rodríguez, 1–13. Washington, DC: The Society for Modeling & Simulation International (SCS).

Wainer, G., R. Goldstein, and A. Khan. 2018. "Introduction to the Discrete Event System Specification Formalism and Its Application for Modeling and Simulating Cyber-Physical Systems". In *2018 Winter Simulation Conference (WSC)*, 177–191 https://doi.org/10.1109/WSC.2018.8632408.

Wooley, A., D. Silva, and J. Bitencourt. 2023, August. "When is a Simulation a Digital Twin? A Systematic Literature Review". *Manufacturing Letters* 35:940–951.

Yao, J.-F., Y. Yang, X.-C. Wang, and X.-P. Zhang. 2023. "Systematic Review of Digital Twin Technology and Applications". *Visual computing for industry, biomedicine, and art* 6:10.

Zeigler, B., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event Iterative System Computational Foundations*. 3rd ed. New York: Academic Press.

Zeigler, B. P. 1976. *Theory of Modeling and Simulation*. New York: John Wiley.

AUTHOR BIOGRAPHY

ARNIS LEKTAUERS is an Associate Professor in the Faculty of Computer Science, Information Technology, and Energy at Riga Technical University, Latvia. He obtained his Ph.D. from Riga Technical University in 2008. His main scientific interests include the research of high-performance interactive hybrid simulation solutions with an application to complex systems analysis in the area of industrial, economic, ecological and sustainable development. He is actively involved in academic and professional activities in various capacities. His email address is arnis.lektauers@rtu.lv.