Modeling and Simulating IoT Infrastructures

Philipp Zech[1][a], Karthik Vaidhyanathan[2][b], Likhith Kanigolla[2][c], Luca Rahm[1] and Ruth Breu[1][d]
[1]*Department of Computer Science, University of Innsbruck, Technikerstrasse 21a, Innsbruck, Austria*
[2]*Software Engineering Research Center, IIIT Hyderabad, Gachibowli, India*

Keywords: Domain-Specific Modeling, Model-Driven Development, DEVS, Model-Driven Simulation, Model-to-Text Generation.

Abstract: Effective management and optimization of urban infrastructures necessitates scalable and accessible simulation frameworks. Modern BIM-based solutions present a promising avenue for novel construction projects; however, these solutions are often inapplicable to the extensive array of legacy infrastructures developed prior to the establishment of BIM as a construction standard. Commensurate with this, we present a new model-driven simulation approach for urban infrastructures that (i) utilizes a novel domain-specific modeling language (DSML) to represent both structural and behavioral characteristics of these infrastructures and (ii) employs the Discrete Event System Specification (DEVS) formalism for simulation purposes. Reframing urban infrastructures as IoT-based, event-driven systems facilitates efficient, hierarchical simulations of complex dynamic environments, including resource management and water networks. Simulation artifacts produced from the DSML through model-to-text generation are executed within the DEVS simulation framework. We validate our approach through a case study conducted at IIIT Hyderabad's Smart City Living Lab, illustrating its capacity to identify optimization opportunities within urban infrastructures.

1 INTRODUCTION

Urban and civil infrastructures consume over two-thirds of the world's energy (International Energy Association, 2021). This has led to the rise of smart infrastructures (Snoonian, 2003), i.e., urban infrastructures equipped with numerous IoT devices for improved monitoring and more timely reaction w.r.t. optimizing, e.g., the operational footprint (King and Perry, 2017). However, this only solves one part of the problem. The situation is more critical if taking a closer look at the large stock of legacy infrastructures. According to the EU's Energy Performance of Building's Directive, 35 million building units are to be renovated to achieve higher resource efficiency (Council of European Union, 2024). Similarly, with the ECO Niwas Samhita 2018, India is also pushing novel building codes to improve buildings' and infrastructures' operational performance (Bureau of Energy Efficiency, Govt. of India, 2018). To even come close to achieving these ambitious goals, accessible, efficient and scalable solutions for the identification of quick optimization potentials of *legacy* infrastructures are much-needed. Modeling and simulation provide an efficient remedy.

Over the last decades, Building Information Modeling (BIM) has emerged as a new, de-facto standardized workflow for digital planning, design, construction, and operation in the architecture, engineering, and construction (AEC) industry (MarketsandMarkets, 2024). This increased adoption naturally has led to the exploration of BIM-based simulations which became feasible as to a collaborative process that integrates digital representations of an infrastructure's physical and functional characteristics to support its design, construction, and operation (Eastman et al., 2011; Di Biccari et al., 2022). However, in the event of legacy infrastructures, as to the recency of BIM, such models largely do not exist thus rendering BIM-based simulations infeasible.

In light of the above, we identify a paramount demand for accessible, efficient and scalable solutions for the identification of quick optimization potentials of legacy infrastructures that does *not* capitalize on BIM. By reframing an infrastructure as a *discrete event system* (Lu and Olofsson, 2014) we even-

[a] https://orcid.org/0000-0002-4952-4337
[b] https://orcid.org/0000-0003-2317-6175
[c] https://orcid.org/0000-0003-2317-6175
[d] https://orcid.org/0000-0001-7093-4341

Zech, P., Vaidhyanathan, K., Kanigolla, L., Rahm, L. and Breu, R.
Modeling and Simulating IoT Infrastructures.
DOI: 10.5220/0013557400003970
In *Proceedings of the 15th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2025)*, pages 93-105
ISBN: 978-989-758-759-7; ISSN: 2184-2841

tually arrive at a feasible and efficient representation, e.g., in terms of its embedded IoT devices, which can be leveraged for discrete event simulation (Agalianos et al., 2020) using the DEVS formalism (Zeigler, 2000). This then directly yields the central research questions investigated in our work, viz.

- **RQ1.** How can we efficiently model infrastructures by their embedded IoT devices to simulate and understand their operational behavior?
- **RQ2.** How can DEVS be leveraged for model-driven, discrete event-based simulations of connected IoT infrastructures for the identification of quick optimization potentials?

Commensurate with the above, in this paper, we present a novel domain-specific modeling language (DSML) for modeling infrastructures at the IoT layer. This allows for capturing both structural and operational behavior aspects in a accessible and efficient way without the need for a BIM model. From the resulting DSML-based models, we generate runtime artifacts as Python code which are then executed in the Python PDEVS simulation environment (Van Tendeloo and Vangheluwe, 2016). This results in an efficient, model-driven workflow that capitalizes on model-based tool integration (Kapsammer et al., 2006) for accessible model-driven simulation of infrastructures for the identification of quick optimization potentials. We evaluate our proposal in a Living Lab at the IIIT Hyderabad in India.

2 BACKGROUND AND RESEARCH PROBLEM

To reduce the global resource footprint of infrastructures, efficient and scalable solutions for the fast identification of optimization potentials in legacy infrastructures are necessary. Unfortunately however, modern BIM-based simulations of legacy infrastructures usually are not feasible as to various reasons:

- *Data Acquisition and Model Creation:* Legacy infrastructures often lack up-to-date or accurate documentation, making it difficult to gather necessary data for BIM. Technologies like laser scanning (LiDAR) or photogrammetry are often used to capture data, but this can be time-consuming and costly and only reconstructs an infrastructure's hull (e.g., a visual 3D model of an infrastructure's manifestation) (Volk et al., 2014).
- *Complexity of Building Forms:* Legacy infrastructures often feature non-standard geometries or irregular forms that are difficult to capture and represent in modern BIM software, which is typically optimized for contemporary designs with regular shapes (Baik et al., 2014).
- *Material Degradation and Unknown Conditions:* Legacy infrastructures often experience material degradation or undocumented changes over time, such as renovations or repairs, which complicate the process of creating accurate BIM models without comprehensive manual examination (Fai et al., 2011).
- *Interoperability of Data:* Many legacy infrastructures only have outdated records in non-digital formats (e.g., paper blueprints or 2D CAD drawings), which are difficult to translate into BIM models without extensive effort. Ensuring interoperability between these formats and BIM software adds to the complexity (Murphy et al., 2013).
- *Cost and Skill Requirements:* The cost of applying BIM to legacy infrastructures is often high due to the need for advanced scanning tools and specialized expertise to interpret the data and translate it into a usable BIM model. The return on investment is uncertain, particularly for non-commercial projects like historic preservation (Becerik-Gerber et al., 2012).
- *Heritage Conservation Concerns:* When BIM is applied to historically significant buildings, it may conflict with heritage conservation principles, which prioritize maintaining the building's original condition. BIM's focus on efficiency and standardization can clash with these values (Dore and Murphy, 2012).

Commensurate with this, we thus look for feasible approaches elsewhere, mainly in the field of DEVS by following the idea of reframing an infrastructure as an event-driven system, where state changes occur at discrete points in time in response to events to enable efficient and hierarchical simulation of complex dynamic systems, e.g., infrastructures (Zeigler, 2000; Lu and Olofsson, 2014; Agalianos et al., 2020).

2.1 Infrastructures as DEVS-Based Simulations

We rely on the fundamental assumption of being able to represent an infrastructure as a DEVS system. Following Lu and Olofsson (2014), a building can be treated as a DEVS system by modeling its components (e.g., sensors and actuators) as atomic models with discrete states that change in response to specific events, such as occupancy detection or temperature fluctuations. These components can be organized hierarchically to represent an IoT infrastructure, with

subsystems (for rooms or floors) interacting through a coupled model that simulates coordinated responses to events, such as adjusting lighting and HVAC when occupants enter a room. DEVS manages time-driven events efficiently, simulating both scheduled operations (e.g., daily heating routines) and unpredictable occurrences (e.g., sudden temperature drops or water contamination). Such an approach enables real-time simulations of dynamic behaviors like energy use or water demand, providing a comprehensive model of infrastructure operations.

Albataineh and Jarrah (2021) developed a DEVS-based framework to evaluate smart home IoT systems, where device scheduling simulations provide actionable recommendations to optimize resource use. Similarly, Antoine-Santoni et al. (2007) applied DEVS formalism to model wireless sensor networks for environmental monitoring, particularly in wildfire detection, showcasing the modular capacity of DEVS in managing complex environmental interactions. Esteban et al. (2023)proposed an early warning system for harmful algal blooms, using DEVS within an IoT architecture to integrate layered simulation and validation stages for real-time monitoring and data processing. Mittal and Martin (2013a) introduced the DEVS Unified Process (DUNIP) for netcentric systems, aligning DEVS with model-driven systems engineering (MDSE) to integrate Service-Oriented Architecture (SOA) and Event-Driven Architecture (EDA), enhancing interoperability for large-scale systems.

Expanding on IoT applications within DEVS, Albataineh and Jarrah (2019) further explored smart home device management by simulating behaviors and interactions among monitoring and control devices. Their DEVS-based IoT management system improves device deployment efficiency and allows simulation of scenarios involving power consumption and cost, providing insights for optimal smart home configuration.

Fazel and Wainer (2023) investigated model-based development for the IoT, focusing on a DEVS-based methodology to streamline IoT application life cycle management through abstraction and reusability. Their work emphasizes the importance of network protocols like MQTT for data transmission, implementing DEVS-based MQTT models to ensure quality of service (QoS) and reliability in connected systems, which are critical for real-time and distributed IoT applications in complex environments like healthcare and industrial settings.

Lastly, Im et al. (2021) introduced methods to simulate mobile IoT systems using DEVS, overcoming challenges associated with simulating dynamic IoT nodes. Their approach incorporates hierarchical partitioning and a novel event-driven technique to manage node mobility, significantly reducing simulation time while maintaining model accuracy. This advancement is especially relevant for applications where IoT devices are mobile, such as in smart city or vehicular networks, addressing scalability and efficiency concerns inherent in large-scale, mobile IoT systems.

Together, these works underscore the versatility of DEVS in modeling, simulation, and optimization for connected, dynamic systems across smart infrastructure applications. However, these works also demonstrate that so far the application of model-driven engineering for simulation engineering to reduce the complexity in capturing an infrastructure's structure and interactions has not been sufficiently explored. Specifically, despite employing a model-based methodology, model-driven engineering's full potential by automation, consistency, validation & verification, simulation generation, and tool integration for faster, scalable development and adaptability to changing system requirements or manifestations remains largely unrealized (Domingo et al., 2020; Cetinkaya et al., 2011; Mittal and Martin, 2013b; Zeigler et al., 2018).

2.2 Case Study: Smart City Living Lab at IIIT Hyderabad

In the following we discuss the ideas from Section 2.1 in the context of our running case study, i.e., the simulation of connected infrastructures in the context of the Smart City Living Lab at the IIIT Hyderabad and its implications for the successful application of DEVS, alongside a brief primer on DEVS.

The Smart City Living Lab at [1] at IIIT Hyderabad's Smart City Research Center is an advanced urban innovation ecosystem, created with support from MEITY (Government of India), the Telangana government, EBTC, and Amsterdam Innovation Arena. Spanning a 70-acre smart campus, it incorporates a distributed IoT infrastructure comprising sensors, actuators, and data systems to address critical urban challenges like air quality monitoring, energy efficiency, weather forecasting, water management, and adaptive smart infrastructure. The lab serves as a real-time testbed for developing and deploying scalable smart city solutions in a live environment, housing over 4,000 residents, research labs, and more than 22 research centers - six of which are directly focused on smart cities.

In our study, the Living Lab offers a rich set of

[1] https://smartcitylivinglab.iiit.ac.in

real-world scenarios (cf. Section 2.4) in a connected infrastructure, which we aim to model and simulate in part using our proposed formalisms. By comparing real system data with the simulation results, we can assess the accuracy and reliability of our models. This comparison enables us to determine if the models effectively represent the underlying system structure and dynamics with sufficient detail, ensuring that the simulations provide a trustworthy basis for informed decision-making. The Living Lab's real-world testing and data collection, particularly in the areas of smart energy, water management, and urban infrastructure, offer valuable insights into the fidelity of our models.

2.3 A Brief Primer on DEVS

An atomic DEVS model is defined as

$$aDEVS = \langle S, ta, \delta_{int}, X, \delta_{ext}, Y, \lambda \rangle \quad (1)$$

with S denoting the set of admissible, sequential states, ta the time advance function, δ_{int} the internal transition function, X the set of admissible, external inputs, δ_{ext} the external transition function for capturing external influences, Y the set of admissible (external) outputs, and λ the output function for mapping internal states onto the output set. The time base T is continuous, e.g., $T = \mathbb{R}$.

To capture interaction among different systems (cf. devices, or atomic models), DEVS further defines a coupled model as

$$cDEVS = \langle X_{self}, Y_{self}, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select \rangle \quad (2)$$

with X_{self} the set of allowed external inputs and Y_{self} the set of allowed (external) outputs. D denotes the set of unique component references (names), $\{M_i\}$ denotes the coupled components with each M_i denoting an $aDEVS$ model (cf. Equation 1), I_i the set of *influencees* of a component M_i, i.e., the components influenced by $i \in D$, thereby specifying the coupling network structure. The set $\{Z_{i,j}\}$ denotes the output-to-input translation functions to map the output from one component $i \in D$ to the input of another $j \in D$. Finally, *select* specifies a tie-breaking between simultaneous events by choosing a unique component from any non-empty subset $E \in D$ where E corresponds to the set of all components having a state transition simultaneously. A more extensive treatment is available from Zeigler (2000).

2.4 Case Study Description

An IoT-driven system has been deployed to optimize water resource management and device monitoring in the Living Lab at IIIT Hyderabad. The aim is to ensure sustainable and efficient urban water usage while maintaining infrastructure responsiveness. The deployed solution seamlessly combines sensors, controllers, and a central gateway (oneM2M (Gezer and Taşkın, 2016)) for real-time data exchange and automated responses among connected components (cf. Figure 1).

At its core, the system uses interconnected nodes to monitor water quality, flow, and levels, while also enabling device control and energy tracking. Esp32 controllers collect data such as pH, turbidity, temperature, water levels, flow rates, and pressure, transmitting this information via HTTP/HTTPS through the oneM2M communication layer for provisioning this data to downstream applications. Designed for interoperability and scalability, this system not only addresses immediate urban water challenges but also lays a robust foundation for integrating additional IoT capabilities in the future.

This scenario is valuable to our work as it highlights a single node within a connected infrastructure (e.g., the Living Lab), composed of interlinked subcomponents like sensors and actuators. These components communicate either directly or through the oneM2M layer. The modular design supports easy adaptation to other applications by exchanging components. Moreover, the scenario exemplifies the modular and hierarchical structure of such systems. Consequently, by adopting a modular and hierarchical modeling approach that allows to capture such structures, our proposed solution readily achieves scalability in model size.

2.5 Challenges and Contributions

Commensurate with our discussions so far and in light of the research questions from Section 1, we identify the following challenges, viz.:

1. How to model infrastructures at the IoT-device level?
2. How to model operational behavior of such infrastructure?
3. How to generate a DEVS simulation from the infrastructure model?

Aligned with these, we deliver the following contributions:

1. A DSML for modeling both
   - infrastructures as connected IoT networks, and
   - their accompanying operational behavior for DEVS-based simulation.

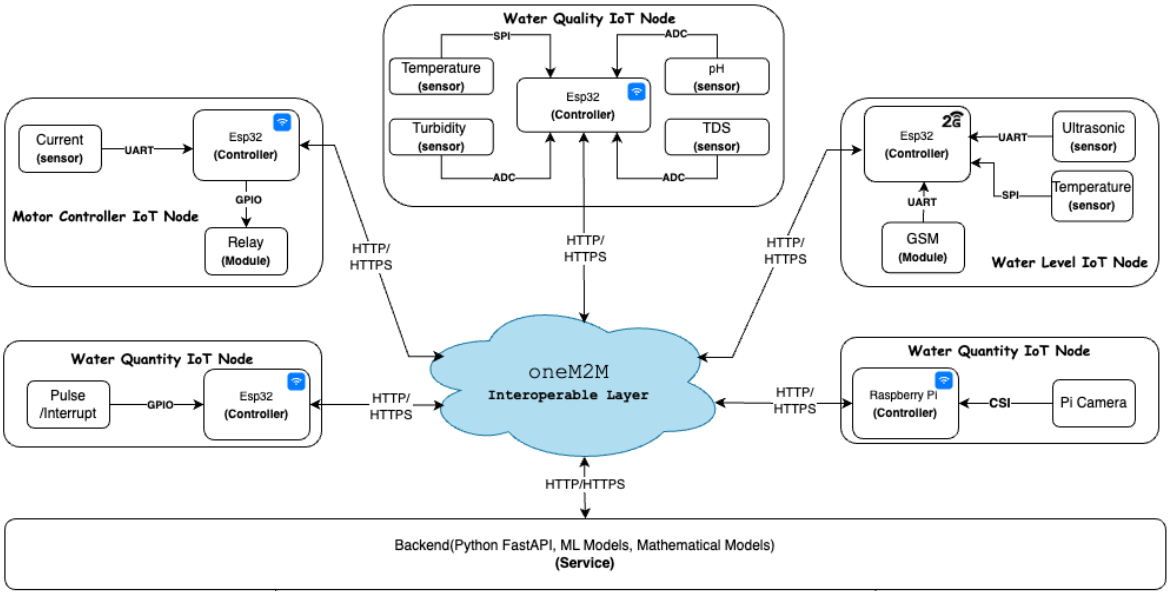

Figure 1: Water quality control scenario: The diagram illustrates the data flow between IoT nodes, communication protocols (HTTP/HTTPS), and the oneM2M gateway, which enables seamless interoperability. Sensor data from nodes measuring water quality, levels, flow, and device status is transmitted to a centralized back-end for processing and visualization, forming the backbone of the water management infrastructure.

2. Model-2-text code generators for emitting both *aDEVS* and *cDEVS* models (and accompanying boilerplate code) for simulation using Python PDEVS (Van Tendeloo and Vangheluwe, 2016).

3 MODEL-DRIVEN SIMULATION OF CONNECTED INFRASTRUCTURES

Our work leverages earlier results by Albataineh and Jarrah (2019, 2021), Im et al. (2021), Esteban et al. (2023), and Fazel and Wainer (2023) who introduced the conceptual foundations on simulating IoT devices using DEVS. Our work builds atop their results by introducing a DSML for modeling IoT devices and their infrastructure topology for the subsequent automated generation of executable Python PDEVS simulations. In the following we outline our modeling infrastructure, the metamodel of our DSML for modeling connected IoT infrastructures, and finally elaborate on the model-2-text generation of executable DEVS simulations using Python PDEVS.

Scope: The modeling methodology covers the modeling of infrastructures at the IoT-level which includes the definition of devices and their connections. In addition, operational behavior and environmental properties like transmission delay or the frequency of sensors readings is also included as part of the modeling process.

Modeling Languages: Eclipse Xtext (Bettini, 2016) is used for metamodelling, i.e., defining the abstract and concrete syntax of our DSML. This yields the necessary tooling for modeling and later code generation as an Eclipse plugin (d'Anjou, 2005). Code generation is done using Eclipse Xtend (Bettini, 2016; Birken, 2014).

Runtime Infrastructure: Python PDEVS (Van Tendeloo and Vangheluwe, 2016) is leveraged for running DEVS simulations. The relevant artifacts, i.e., executable Python code yield from an Xtend-based model-2-text generator (cf. Section 3.3).

3.1 A DSML for Modeling Connected IoT Infrastructures

The metamodel outlined in Figure 2 delineates a detailed schema that governs the interaction and functionality of diverse devices (as to space restrictions, `enum` types such as `LinkType` or `DataType` are not shown). At the core of this model is the `CommunicationLink`, which functions as a conduit enabling uninterrupted communication between devices. Each instance of `CommunicationLink` is characterized by several essential attributes: an `id`, which serves as a unique string identifier representing the specific link instance (inherited from `Linkable`, see below); a `protocol` that describes the communication methodology – options include `HTTPS`, `UART`, among

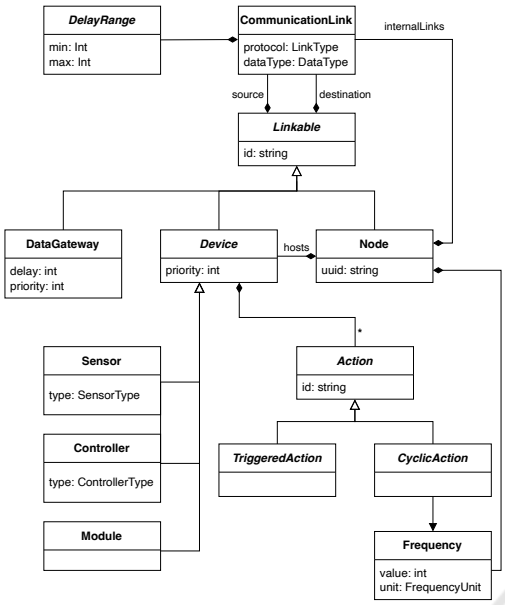

Figure 2: Metamodel for modeling connected IoT infrastructures. As to space restriction, enum types are not shown.

others; a dataType, which specifies the nature of the data being transmitted (e.g., Boolean values, floating-point numbers).

Complementary to the CommunicationLink, the metamodel incorporates a Linkable entity, which denotes any device that possesses the capability to interface via the CommunicationLink. This entity facilitates the heterogeneous integration of devices, thereby ensuring effective communication among various components within the ecosystem. The metamodel encompasses a variety of SensorType entities, such as PH, TURBIDITY, TEMPERATURE, among others, each specifically designed to monitor distinct environmental conditions that are essential for effective building management. Simultaneously, Controller entities, including ESP32 and RASPBERRY_PI, represent the computational architecture that governs these sensors, thereby enabling real-time data processing and decision-making capabilities. Our DSML avoids the need for a dedicated actuator entity by capturing actionable behavior inside Devices (see below).

To facilitate precise regulation of the operations conducted by the devices, the DSML integrates Frequency, which models the temporal intervals for the execution of actions or the monitoring of tasks. Units may encompass HERTZ, SECONDS, MINUTES, among others, facilitating the system's adaptation to diverse operational requirements.

The DataGateway possesses the capability to manage various data types, including BOOL and FLOAT,thereby offering flexibility in the administration of diverse device outputs. The interoperability is further enhanced by attributes such as priority, which determines the urgency of actions; delay, which specifies the latency prior to executing an action. In practice, this DataGateway can be realized by various implementations, e.g., a data store or an external API to forward data to an external consumer. As such external actors however are not the core focus of our work, the DataGateway is implemented as a simple data store (cf. Section 4.2).

Alongside these fundamental components, the model delineates a range of actions that devices are capable of executing, which are classified under the TriggeredAction and CyclicAction entities. Each action is subject to configuration through parameters including min and max thresholds, postingFrequency, and a distinct id. Collectively, this facilitates precise regulation of operational behavior, guaranteeing that devices react appropriately to both internal and external inputs and directives.

Observe that the DataGateway does not inherit from Device as the former is not intended to be a part of the latter, i.e., DataGateways exist outside Nodes.

Our metamodel establishes a comprehensive framework for the representation and management of IoT devices within smart building environments. This facilitates organized communication, promotes interoperability among various devices, and permits adaptive response mechanisms customized to the specific requirements and functionalities of interconnected systems. Consequently, the model facilitates the primary objective of developing intelligent, responsive environments that improve operational efficiency and user experience.

3.2 Modeling Steps

The modeling procedure for connected IoT infrastructures for DEVS simulations comprises five steps:

(1) Defining IoT devices in the model, including their types (e.g., sensors, actuators, or controllers) and configurations, (2) Establishing device connections and configurations, (3) Specifying operational behavior and properties, (4) Code generation, and (5) Validation and testing. Crucially, the below steps need to be followed in order.

(1) Defining IoT Devices. To begin, IoT devices (e.g., sensors or controllers) are modeled through the utilization of the provided language constructs (e.g., Sensor or Controller) as defined in by metamodel (cf. Figure 2). This includes the definition of relevant devices properties for later generating complete simulation artifacts (see (3) below). Listing 1, among

others, shows devices as modeled as part of the scenario from Figure 1.

(2) Establishing Device Connections. Moving on from modeling the IoT devices, the second step involves creating a topology by connecting the various modeled devices with each other using the `CommunicationLink` construct. This results in the establishment of a *connected infrastructure* for subsequent simulation and analysis. As for devices, this also comprises the modeling of connection properties, i.e., the underlying `Protocol` or a `Delay`. Listing 1, among others, shows various modeled connections between modeled devices as part of implementing the scenario from Figure 1.

(3) Specifying Operational Behavior and Properties. This step focuses on establishing parameters that can influence the behavior of IoT devices, including factors such as transmission delays or sensor reading frequencies. By carefully defining these properties, users are empowered to simulate diverse scenarios that closely mirror real-world conditions, thereby improving the realism and reliability of the resulting simulations (cf. Listing 1 where we model the `postFrequency` for nodes).

(4) Code Generation. With the model fully established, in the next step, executable simulation artifacts are generated for Python PDEVS. Aside from generating the necessary *aDEVS* and *cDEVS* models (cf. Equations 1 and 2) this also comprises the generation of relevant boilerplate code for running simulations (cf. Listing 2) or *Generators* (Van Tendeloo and Vangheluwe, 2016) for a sensor to actually send values during simulation (cf. Figure 4). Observe that at this point, relevant configuration data from modeled entities is leveraged, e.g., a node's `Frequency` in the *ta* function of a its corresponding *aDEVS* model. Section 3.3 further elaborates on our underlying model-2-text generator.

(5) Validation and Testing. The final step entails validating the generated model to confirm that it accurately reflects the intended system and that the generated code performs as expected (cf. Section 4.2). Preliminary simulations can be conducted to evaluate the model's fidelity in comparison to real-world data or established benchmarks. Should any discrepancies arise during this testing phase, they can prompt iterative refinements in device definitions or connection parameters, thereby fostering continuous improvement and optimization of the model. The model-driven nature of our approach naturally facilitates such iterative development (cf. *IterSPEC* (Zeigler, 2000)).

```
node waterQuantityNodeCam {
        sensor cameraWQ type CAMERA priority 1
        controller raspiWQ type RASPBERRY_PI priority 3

        postFrequency 300 SECONDS

        link SPI from cameraWQ -> raspiWQ {
                delay [1..12]
                datatype CAMERA
        }
        priority 3
}

node waterQuantityNodePulse {
        sensor pulseWQ type PULSE priority 1
        controller espWQP type ESP32 priority 3

        postFrequency 300 SECONDS

        link SPI from pulseWQ -> espWQP {
                delay [1..12]
                datatype PULSE
        }
        priority 3
}
```

Listing 1: Model of the `WaterQuantityNodeCam` and `WaterQuantityNodePulse` from Figure 1.

```
model = MyModel()
sim = Simulator(model)
sim.simulate()
```

Listing 2: Python PDEVS boilerplate code for running a simulation, which is also generated as part of our model-2-text code generation.

3.3 Code Generation

Our model-2-text code generator plays a pivotal role in ensuring that the user-defined models are accurately represented in the simulation environment. The algorithm consists of various stages, each aimed at translating specific elements of the DSML into executable code.

As shown in Algorithm 1, for generating and *aDEVS* simulation model for a node, our algorithm begins by defining a state class for the node, which is named with the node's name. Within this state class, a dictionary named `data_aggregated` is initialized to maintain a node's state.

Following this, an instance of an `AtomicDEVS` class is established for the node itself, featuring a constructor that sets relevant properties such as the node's name and pin configuration. This constructor also creates an instance of the state class defined earlier and keeps track of the last execution time via an

Algorithm 1: Compile Node.

Input: Node node with links, frequency, and priority
Output: Atomic DEVS model

Initialize `nodeLinks` (list) and `distinctLinkTypes` (set) from `node.links`;
Compute `postFrequency` from `node.freqValue` and `node.freqUnit`;
Determine `knownController` based on `node.controller` type;
Generate Class `NodeState` definition:
    Initialize `data_aggregated` (empty) and set `next_internal_time` with randomness around `postFrequency`;
Generate Class `Node (AtomicDEVS)` **definition**:
    Initialize `(name, pinout)`:
        Set `state` (NodeState), `timeLast` (0.0), `pins` (pinout);
        Create input ports for `distinctLinkTypes`;
        Define output port and set node priority;
    Generate Function `timeAdvance()`:
        Return time until next event if data exists, else `INFINITY`;
    Generate Function `extTransition(inputs)`:
        Update state with received inputs if matching links exist;
        Update `timeLast` to `next_internal_time`;
    Generate Function `intTransition()`:
        Increment `next_internal_time` and update `timeLast`;
    Generate Function `outputFunc()`:
        If data exists, package with timestamp, send via output, and clear state;

```python
class PulseWQ(AtomicDEVS):
    def __init__(self, data_generator):
        super().__init__('pulseWQ')
        self.inport = self.addInPort("in_port")
        self.outport = self.addOutPort("outport")
        self.state = {'PULSE': 0}
        self.priority = 1
        self.gen = data_generator

    def intTransition(self):
        self.state['PULSE'] = self.gen.value('pulse')
        return self.state

    def extTransition(self, inputs):
        return self.state

    def outputFnc(self):
        return {self.outport: self.state['PULSE']}

    def timeAdvance(self):
        return 1.0

    def __lt__(self, other):
        return self.priority < other.priority
```

Listing 3: Generated atomic DEVS model for the pulse sensor from Listing 1.

internal variable called `timeLast`. In addition, this constructor dynamically generates input ports for different link types that the node will manage.

The primary functionalities of a `Node` are defined in several methods as required by the DEVS formalism (cf. Section 2):

- `timeAdvance` determines the duration until the next internal transition, returning `INFINITY` if there is no aggregated data.
- `extTransition` oversees external transitions triggered by inputs, executing specific logic upon re-

```python
class DataGenerator:
    def __init__(self,
                 mode='random',
                 config='generator/sensors_config.json',
                 csv_file=None):
        if mode == 'csv' and csv_file is None:
            raise ValueError('CSV mode: Invalid CSV file')
        elif mode == 'csv':
            self.generator = CSVDataGenerator(csv_file)
        elif mode == 'random':
            self.generator = RandomDataGenerator(config)
        else:
            raise ValueError(f"Unsupported mode: {mode}")

    def value(self, sensor_name):
        return self.generator.generate_value(sensor_name)

    def pulse_value(self, sensor_name):
        return self.generator.pulse_value(sensor_name)

    def camera_value(self, name):
        return self.generator.camera_value(sensor_name)
```

Listing 4: Generated `Generator` for the pulse sensor model from Figure 3.

ceiving data from a designated controller.

- `intTransition` updates the internal state according to the current conditions of the node and increments the `next_internal_time`.
- `outputFunc` formats the output data to be sent, which includes a timestamp and the data that has been aggregated.

Generation for other entities (e.g., `CommunicationLink` or `DataGateway`) follows the same procedure, i.e., a template-based approach where modeled entities are expanded into executable *aDEVS* models. The full algorithm of our code generator is available for download from https://doi.org/10.5281/zenodo.15172097. Figures 3 and 4 show examples of generated and readily executable Python PDEVS code utilizing the generator from Figure 4 on the model from in Listing 1.

Omitted in Algorithm 1 due to spatial constraints, our code generator additionally produces the *cDEVS* model to ultimately simulate all the nodes depicted in Figure 1 (see Section 4).

By relying on Eclipse Xtend and our translation algorithm, we ensure efficient and scalable generation of simulation models, facilitating rapid prototyping and analysis of connected IoT infrastructures. This streamlined approach significantly reduces the complexity traditionally associated with model development, thereby enhancing usability for researchers and practitioners alike by facilitating the efficient generation of simulation-ready code but also supports quick modifications and testing based on the defined model.

4 RESULTS AND DISCUSSION

In the following, we discuss (i) the modeling of the scenario from Figure 1 (cf. Section 4.1), (ii) running simulations of the modeled scenario (cf. Section 4.2), and (iii) the results retrieved from the simulation (cf. Section 4.3) with the developed model in Figure 1. We conclude this section with a discussion of our results (cf. Section 4.4).

Our primary focus is on (a) ensuring completeness and correctness in relation to the guiding scenario, (b) assessing the engineering effort needed to develop working simulations within our proposal, and (c) evaluating the implications of our approach for adopting end users. Accordingly, we will not provide an in-depth analysis of our DSML which will be evaluated as part of future work using the Technology Acceptance Model (Davis et al., 1989).

4.1 Modeling the Living Lab

We have modeled the scenario from Figure 1 following the steps outlined in Section 3.2. Consequently, the modeling process followed a structured approach to develop a simulation-ready representation of the scenario's IoT infrastructure. This involved defining components, establishing interactions, and generating executable DEVS simulation artifacts in four steps:

1. *Defining IoT Devices*: Key IoT components, including sensors (water quality, temperature, flow), controllers (ESP32, Raspberry Pi), and actuators, were modeled with properties like data type, communication protocol, priority, and data transmission frequency (cf. Lst. Listing 1). The oneM2M component is modeled by the `DataGateway` as a simple data sink.
2. *Establishing Device Connections*: Device interactions were defined using `CommunicationLink`, representing protocols such as HTTP, SPI, and UART. Each connection included parameters like transmission delay and data type to reflect real-time data exchange (cf. Lst. Listing 1).
3. *Specifying Operational Behavior and Properties*: Sensor reading frequencies, transmission delays, and environmental conditions were incorporated to ensure realistic behavior. For example, a water quality sensor transmitting data every 300 seconds was modeled with corresponding frequency and latency (cf. Lst. Listing 1).
4. *Code Generation*: Our model-to-text generator (cf. Section 3.3) generated Python PDEVS simulation artifacts, including *aDEVS* and *cDEVS* models, and accompanying *Generators*. A 75-line input model produced 740 lines of Python PDEVS code. Scaling the model resulted in a sublinear increase in code generation time, confirming efficiency (cf. Table 1). In addition, these results also show the high expressiveness of our proposed DSML by capturing and abstracting away tedious low-level details during simulation modeling.

As to space restrictions, we cannot show the complete model for the scenarios from Listing 1; however, it is available from our artifact.

Our structured modeling approach enables the rapid transformation of a conceptual IoT infrastructure into a functional simulation. Using domain-specific modeling and automated code generation significantly reduces complexity and manual effort in setting up large-scale simulations. In addition, as shown by the data from Table 1, our proposed DSML offers a high-level of expressivity by effectively ab-

Table 1: Performance evaluation of the code generator. Generation time is averaged over three runs. Our results show that our proposal achieves scalability in model size by being able to handle substantially large models. *Baseline* refers to the scenario from Figure 1, i.e., the number of entities modeled therein; consequently, *2x* implies double the entities, *4x* four times the entities, etc.

Factor	Input LoC	Generated LoC	Time (ms)
Baseline	75	740	1452
2×	145	1330	1494
4×	285	2510	1553
8×	565	4870	1652
32×	2245	19030	2160

Input LoC: Lines of "code" in the input model; Generated LoC: generated number of lines of code

stracting away low-level modeling details in simulation engineering.

4.2 Simulating the Living Lab

The simulation setup utilized real-world sensor data collected over 12 months (available from our artifact) to train a distributed lognormal model (Crow and Shimizu, 1987), which served as the basis for synthetic data generation. The goal was to validate the accuracy of the modeled infrastructure and assess its performance under different conditions. The trained lognormal distribution produced simulated sensor values that were tested in real-time scenarios, enabling direct comparison with live measurements. The generated DEVS models were executed using Python PDEVS, following a structured workflow:

1. Initialization of the simulation environment with the Living Lab's IoT components.
2. Execution of time-stepped simulations to model sensor data collection, device interactions, and real-time responses.
3. Comparison of simulated outputs with real-world measurements to ensure model fidelity.

The simulation successfully reproduced observed behaviors, such as water quality and water flow rate fluctuations and automated actuator responses. The results validated through repeated observations and closely matched real-world data, confirm the model's reliability. For example, Figure 3 shows a comparison of real-world water quality and water flow rate readings compared to data collected during simulation. As clearly visible from Figure 3, our simulations are able to reproduce real-world behavior accurately.

The simulation validated the accuracy and efficiency of the developed model (cf. Section 4.1). By replicating real-world behaviors and supporting scenario testing, our proposal provides a valuable tool for optimizing IoT-based infrastructure operations. Observe that this validations comprises step 5 of our modeling methodology (cf. Section 3.2).

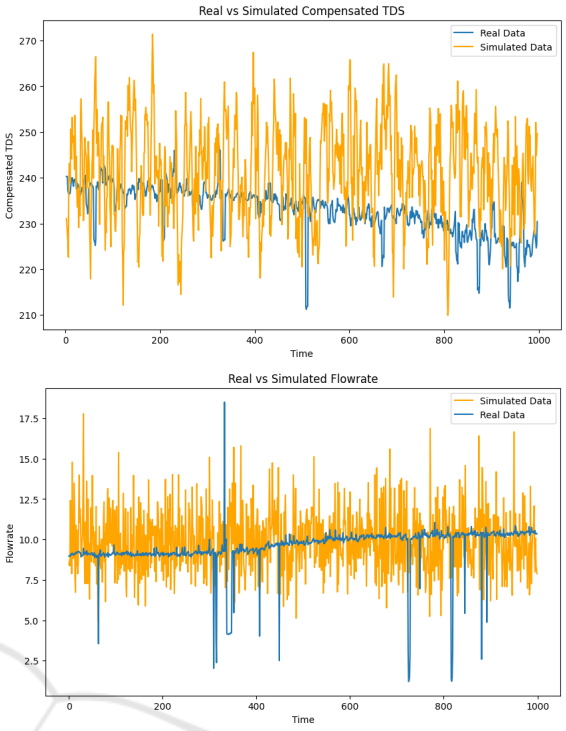

Figure 3: Comparison of real and simulated data for Compensated TDS and Flowrate. The graph shows 1,000 data points, reflecting both real and simulated values across multiple observation cycles.

4.3 Results

The simulation fidelity of our approach was evaluated by analyzing its ability to replicate the actual data of the water quality network. Key metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are used to evaluate alignment between simulated outputs and observed values. As we can observe from Figure 3, for compensated TDS, the MAE value is 29.83844 denoting a reasonable simulation fidelity, while for Flow Rate, the MAE value is 1.686508 denoting a high simulation fidelity for the generated data. This is further reinforced by the RMSE score of 37.09545 for Compensated TDS and 2.204044 for Flow Rate between the actual values and simulated values. There are instances where the simulated values deviate from actual values, particularly in situations when there is variation in steady-state flow. This can be potentially improved by improving the granularity of modeling and simulation but this is beyond the scope of this work.

Overall, as can be seen, the results demonstrate

Table 2: Summary of simulation results.

Metric	Compensated TDS	Flow Rate
MAE	29.83844	1.686508
RMSE	37.09545	2.204044

a strong correlation between simulated data and observed values, capturing key features such as water flow, water quality rate, etc. Table 2 summarizes the simulation results.

4.4 Discussion

Our study introduced a novel DSML and a model-driven approach leveraging the DEVS formalism to simulate IoT infrastructures. Through the Smart City Living Lab case study, we validated the effectiveness of our approach in accurately modeling, simulating, and analyzing operational behaviors of connected infrastructures. The results support our research questions while offering insights into both the capabilities and limitations of our proposal.

In response to RQ1, we demonstrated how connected infrastructures can be efficiently modeled at the IoT-device level using our DSML. By encapsulating both structural and operational aspects, our language facilitates a systematic representation of infrastructure dynamics without relying on the availability of BIM models. The modeling process—spanning device definitions, communication links, and behavioral properties—proved adaptable and expressive, capturing complex dependencies within the IoT ecosystem. The automated model-to-text code generation further reduced manual effort, accelerating the transition from conceptual models to executable simulations.

Regarding RQ2, our results affirm that DEVS is a suitable foundation for model-driven, discrete event-based simulations of connected IoT infrastructures. By utilizing Python PDEVS, we achieved an efficient simulation environment capable of handling large-scale infrastructures. The comparison between real-world data and simulated outputs exhibited a reasonable degree of fidelity, particularly for flow rate and water quality metrics. This suggests that our DEVS-based approach can deliver actionable insights for optimizing resource management and infrastructure performance.

Our model-driven workflow proved to be both scalable and adaptable. The ability to generate complex simulation artifacts from concise, high-level specifications enhances both productivity and model accuracy. Performance measurements demonstrated that the code generation scales sublinear with model complexity, making our approach viable for larger infrastructures. Furthermore, the structured methodology supports iterative refinement and validation, a critical feature for real-world applications.

Nevertheless, certain limitations persist. The granularity of our models, while sufficient for our case study, may not capture more intricate physical phenomena or emergent behaviors in highly dynamic environments. Additionally, while the DSML abstracts much of the modeling complexity, the reliance on accurate input data remains a bottleneck for generalization. It should be emphasized, however, that this does not affect the fidelity of a simulation as evidenced by the results (cf. Section 4.2 and Section 4.3), but rather pertains to the correctness of a simulation in general, specifically how closely it mirrors the underlying real-world event. Future work should explore advanced modeling constructs for representing environmental factors and heterogeneous sensor behaviors.

While the simulation accurately replicated real-world behaviors, discrepancies observed in Figure 3 between simulated and actual data underscore the need for further refinement. These variations may stem from model assumptions or limitations in capturing the full complexity of real-world dynamics, suggesting potential areas for improvement in future iterations. To address these discrepancies, we plan to incorporate additional validation metrics and sensitivity analyses, ensuring a more robust assessment of model fidelity.

Furthermore, comparative evaluations with existing simulation frameworks will be conducted to highlight the unique advantages of our approach, particularly in terms of scalability and ease of integration. Looking ahead, we aim to explore the broader applicability of our framework beyond urban water management, potentially extending it to smart energy grids and intelligent transportation systems. This expansion will not only validate the flexibility and adaptability of our approach but also contribute to the development of more efficient and responsive urban environments.

4.4.1 Practical Implications

The findings of our work hold significant practical implications for the management and optimization of urban infrastructures. By providing a model-driven simulation framework that does not rely on BIM, we offer a viable solution for legacy systems that often lack such advanced modeling tools. The capacity to simulate infrastructures at the IoT-level allows urban planners, engineers, and decision-makers to visualize complex interactions within urban environments and identify areas for optimization in real-time. This

could enable more informed resource allocation, improved resource efficiency, and enhanced responsiveness to operational challenges. Additionally, the use of our DSML facilitates quick adjustments and testing of various scenarios without the extensive time and labor often associated with conventional modeling approaches. Ultimately, this research contributes to the development of smart cities by equipping stakeholders with more effective tools for responding to urban challenges, paving the way for sustainable development and improved quality of life in urban areas.

5 CONCLUSIONS

This paper presented a novel DSML and a model-driven simulation approach for IoT infrastructures using the DEVS formalism. We successfully applied this methodology to a real-world Smart City Living Lab, showcasing its ability to capture and simulate operational behaviors. Our findings underscore the value of a model-driven approach in enabling rapid prototyping, iterative validation, and detailed scenario analysis.

The primary contributions of this work are: (i) a DSML tailored for modeling both structural and operational aspects of connected infrastructures, (ii) an automated model-to-text generator for producing Python PDEVS artifacts, and (iii) an empirical validation through an IoT-driven water management system. These contributions collectively advance the state of the art in IoT infrastructure modeling and simulation, particularly for legacy systems where traditional BIM approaches are infeasible.

While promising, several avenues remain for future exploration. First, enhancing the DSML to capture more nuanced operational behaviors—such as user interactions, environmental variability, and material degradation—would improve model fidelity. Another critical direction involves extending the scope of our methodology to additional domains beyond urban water management, including smart energy grids and intelligent transportation systems. In addition, incorporating adaptive simulation techniques and parallelized execution could significantly enhance scalability and simulation responsiveness. Lastly, a comprehensive user-centered evaluation of the DSML's usability and its acceptance is required to understand its broader applicability.

In conclusion, our DSML and DEVS-based framework provide a robust foundation for simulating connected infrastructures. While further research is needed to address scalability and fidelity challenges, our work offers a promising path toward more efficient and accurate modeling of complex, interconnected urban systems.

ACKNOWLEDGMENTS

This research was partly supported by Ministry of Electronics and Information Technology (MeitY), Govt. of India under grant no. 3070665 (2020) as part of Smart City Living lab project and is sustained by corporate funding.

REFERENCES

Agalianos, K., Ponis, S., Aretoulaki, E., Plakas, G., and Efthymiou, O. (2020). Discrete event simulation and digital twins: review and challenges for logistics. *Procedia Manufacturing*, 51:1636–1641.

Albataineh, M. and Jarrah, M. (2019). DEVS-Based IoT Management System for Modeling and Exploring Smart Home Devices. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 73–78. IEEE.

Albataineh, M. and Jarrah, M. (2021). DEVS-IoT: performance evaluation of smart home devices network. *Multimedia Tools and Applications*, 80(11):16857–16885.

Antoine-Santoni, T., Santucci, J. F., De Gentili, E., and Costa, B. (2007). Modelling & simulation oriented components of wireless sensor network using DEVS formalism. In *Proceedings of the 2007 Spring Simulation Multiconference - Volume 2*, page 299–306. Society for Computer Simulation International.

Baik, A., Alitany, A., Boehm, J., and Robson, S. (2014). Jeddah Historical Building Information Modelling" JHBIM"–Object Library. International Society for Photogrammetry and Remote Sensing (ISPRS).

Becerik-Gerber, B., Jazizadeh, F., Li, N., and Calis, G. (2012). Application areas and data requirements for BIM-enabled facilities management. *Journal of construction engineering and management*, 138(3):431–442.

Bettini, L. (2016). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.

Birken, K. (2014). Building code generators for dsls using a partial evaluator for the xtend language. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change: 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part I 6*, pages 407–424. Springer.

Bureau of Energy Efficiency, Govt. of India (2018). Eco Niwas Samhita. urlhttps://www.beepindia.org/eco-niwassamhita/.

Cetinkaya, D., Verbraeck, A., and Seck, M. D. (2011). Mdd4ms: a model driven development framework for modeling and simulation. In *Proceedings of the 2011*

summer computer simulation conference, pages 113–121.

Council of European Union (2024). Council directive (EU) no. 1275/2024. https://energy.ec.europa.eu/topics/energy-efficiency/energy-efficient-buildings/energy-performance-buildings-directive_en.

Crow, E. L. and Shimizu, K. (1987). *Lognormal distributions*. Marcel Dekker New York.

d'Anjou, J. (2005). *The Java developer's guide to Eclipse*. Addison-Wesley Professional.

Davis, F. D. et al. (1989). Technology acceptance model: Tam. *Al-Suqri, MN, Al-Aufi, AS: Information Seeking Behavior and Technology Adoption*, 205(219):5.

Di Biccari, C., Calcerano, F., D'Uffizi, F., Esposito, A., Campari, M., and Gigliarelli, E. (2022). Building information modeling and building performance simulation interoperability: State-of-the-art and trends in current literature. *Advanced Engineering Informatics*, 54:101753.

Domingo, Á., Echeverría, J., Pastor, O., and Cetina, C. (2020). Evaluating the benefits of model-driven development: empirical evaluation paper. In *Advanced Information Systems Engineering: 32nd International Conference, CAiSE 2020, Grenoble, France, June 8–12, 2020, Proceedings 32*, pages 353–367. Springer.

Dore, C. and Murphy, M. (2012). Integration of Historic Building Information Modeling (HBIM) and 3D GIS for recording and managing cultural heritage sites. In *2012 18th International conference on virtual systems and multimedia*, pages 369–376. IEEE.

Eastman, C., Teicholz, P., Sacks, R., and Liston, K. (2011). BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors.

Esteban, S., Risco-Martín, J. L., Chacon, J., Besada-Portas, E., and Andrade, G. A. (2023). Project Simulation, Validation and Deployment with DEVS. IoT Framework for Blooms Monitoring and Alert. In *2023 Winter Simulation Conference (WSC)*, pages 2603–2614. IEEE.

Fai, S., Graham, K., Duckworth, T., Wood, N., and Attar, R. (2011). Building information modelling and heritage documentation. In *Proceedings of the 23rd International Symposium, International Scientific Committee for Documentation of Cultural Heritage (CIPA), Prague, Czech Republic*, pages 12–16.

Fazel, I. A. and Wainer, G. (2023). A DEVS-Based Methodology for Simulation and Model-Driven Development of IoT. In *International Conference on Simulation Tools and Techniques*, pages 3–17. Springer.

Gezer, C. and Taşkın, E. (2016). An overview of onem2m standard. In *2016 24th signal processing and communication application conference (SIU)*, pages 1705–1708. IEEE.

Im, J. H., Oh, H.-R., and Seong, Y. R. (2021). Simulation of a Mobile IoT System Using the DEVS Formalism. *Journal of Information Processing Systems*, 17(1):28–36.

International Energy Association (2021). Global Energy Review 2021.

Kapsammer, E., Reiter, T., and Schwinger, W. (2006). Model-based tool integration-state of the art and future perspectives. In *Proceedings of the 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2006)*.

King, J. and Perry, C. (2017). *Smart buildings: Using smart technology to save energy in existing buildings*. American Council for an Energy-Efficient Economy Washington, DC, USA.

Lu, W. and Olofsson, T. (2014). Building information modeling and discrete event simulation: Towards an integrated framework. *Automation in construction*, 44:73–83.

MarketsandMarkets (2024). Building Information Modeling (BIM) Market – Global Forecast to 2029. Accessed: 2025-02-02.

Mittal, S. and Martin, J. L. R. (2013a). Model-driven systems engineering for netcentric system of systems with DEVS unified process. In *2013 Winter Simulations Conference (WSC)*, pages 1140–1151. IEEE.

Mittal, S. and Martin, J. L. R. (2013b). Model-driven systems engineering for netcentric system of systems with devs unified process. In *2013 Winter Simulations Conference (WSC)*, pages 1140–1151. IEEE.

Murphy, M., McGovern, E., and Pavia, S. (2013). Historic Building Information Modelling–Adding intelligence to laser and image based surveys of European classical architecture. *ISPRS journal of photogrammetry and remote sensing*, 76:89–102.

Snoonian, D. (2003). Smart buildings. *IEEE spectrum*, 40(8):18–23.

Van Tendeloo, Y. and Vangheluwe, H. (2016). An overview of PythonPDEVS. *JDF*, 2016:59–66.

Volk, R., Stengel, J., and Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings—Literature review and future needs. *Automation in construction*, 38:109–127.

Zeigler, B. P. (2000). Theory of modeling and simulation. *Academic Press google schola*, 2:1779–1785.

Zeigler, B. P., Mittal, S., and Traore, M. K. (2018). Mbse with/out simulation: State of the art and way forward. *Systems*, 6(4):40.