Highlights

A Model-Driven Approach for Engineering Mobility Digital Twins: The Bologna Case Study

Alessandra Somma, Domenico Amalfitano, Alessio Bucaioni, Alessandra De Benedictis

- Six key challenges in Mobility Digital Twin development are identified from the literature, including interoperability, adaptability, and automatability.
- A model-driven approach is introduced to structure MoDT development through layered abstractions and automated transformations via the developed M2DT tool.
- The approach is applied to a real case study for the city of Bologna, using real-world traffic data and open urban infrastructure datasets.
- The resulting BoMoDT platform integrates FIWARE and SUMO, supporting simulation, monitoring, and visualization of urban mobility conditions.
- Simulation fidelity and monitoring responsiveness are quantitatively evaluated, confirming BoMoDT's accuracy and effectiveness.

A Model-Driven Approach for Engineering Mobility Digital Twins: The Bologna Case Study

Alessandra Somma[a], Domenico Amalfitano[a], Alessio Bucaioni[b], Alessandra De Benedictis[a]

[a]*University of Naples Federico II, Naples, 80125, Italy*
[b]*Mälardalen University, Västerås, 72220, Sweden*

Abstract

Context. As cities grapple with increasing congestion, sustainability concerns, and the need for efficient mobility systems, *Mobility Digital Twins* (MoDTs) have emerged as promising technology for improving urban transportation. However, the development of MoDTs remains hindered by challenges such as structural complexity, data heterogeneity, lack of interoperability, and limited support for scalability, maintainability, and adaptability.
Objectives. This work aims to address these barriers by introducing a structured and systematic engineering framework that supports the design development of MoDT, reducing technical debt, development costs and human errors, while promoting long-term evolution.
Methods. We propose a *Model-Driven Engineering* (MDE) approach that organizes the development of MoDTs through models at different levels of abstraction and adopts automated transformations from high-level specifications to executable code artifacts, supporting MoDT life-cycle.
Results. The proposed approach is validated through its application in developing a MoDT for the city of Bologna, Italy. To support this, we introduce the **M2DT** tool, which automates the workflow from high-level models to software code artifacts. The resulting **BoMoDT** platform is built using open-source technologies and real mobility data. This case study demonstrates the feasibility and effectiveness of our approach, which, to our knowledge, is the first to apply a model-driven strategy for the entire MoDT development. A qualitative evaluation confirms that our framework addresses key challenges in MoDT development. Quantitative experiments further validate BoMoDT's ability to accurately reproduce and monitor real urban mobility conditions.
Conclusion. The proposed approach offers a solid foundation for addressing MoDT development challenges. By combining automation with structured abstraction, it improves adaptability and maintainability while enabling scalable integration, helping make MoDTs more accessible for future urban system design.

Keywords: Digital Twins, Urban Mobility, Model-Driven Engineering, Model Transformations

1. Introduction

As urban populations continue to grow, cities face increasing challenges related to traffic congestion, environmental sustainability, and the need for efficient transportation systems [1–3]. Addressing these issues requires more than simply expanding infrastructure; it demands intelligent, data-driven urban management that integrates advanced and emerging technologies [2, 4, 5]. In the realm of mobility, researchers and policymakers are increasingly focusing on **smart mobility** solutions [3, 6], which combine traditional transportation modes like buses and trains with innovative approaches such as ride-sharing, car-sharing, and multi-modal networks. By leveraging cutting-edge technologies, these solutions enhance efficiency, safety, and sustainability, creating a more effective urban transportation system.

Among the various technological advancements, *Digital Twin* (DT) has become a key enabler of smart mobility improvements [2, 6]. In the context of urban transportation, this concept is referred to as **Mobility Digital Twin** (MoDT) [2]. A MoDT is a dynamic, continuously updated digital model that represents traffic flow, vehicle interactions and human movement within a city [2, 7]. It integrates real-world data and provides feedback to the physical city, offering insights to human operators or autonomously executing control actions. By harnessing this bidirectional data flow, MoDT facilitates advanced simulation, analysis, and predictive modeling, enabling city planners and mobility providers to optimize traffic management, make data-driven decisions, anticipate congestion, enhance route efficiency, and reduce environmental impact [2, 8]. Furthermore, MoDTs can support sustainability by lowering carbon emissions and fostering scalable, adaptive, and eco-friendly urban mobility systems [4, 7].

Designing and developing MoDTs presents significant challenges due to the complex and hierarchical nature of urban ecosystems, requiring the composition of multiple interconnected DTs representing different aspects of transportation [5].

Email addresses: alessandra.somma@unina.it (Alessandra Somma), domenico.amalfitano@unina.it (Domenico Amalfitano), alessio.bucaioni@mdu.se (Alessio Bucaioni), alessandra.debenedictis@unina.it (Alessandra De Benedictis)

A notable example is *Virtual Singapore*[1], where DTs for public transport, autonomous vehicles, and many others are unified into a single twin system. Moreover, this composite structure introduces interoperability and scalability issues [9], as the management of huge amounts of data, generated from multiple data sources, and their exchange across diverse infrastructures and models remains a major hurdle. MoDTs initiatives like *Smart Dublin*[2] and *Helsinki Smart Region*[3] [10] exemplify these difficulties. Unlike DTs in controlled environments, MoDTs must operate in dynamic and unpredictable urban settings, posing challenges for long-term maintainability. For example, in *New York City*[4], MoDTs have been explored for congestion pricing and traffic forecasting, but frequent infrastructure changes, evolving regulations, and data drift complicate their maintenance. In this context, automated development strategies that enhance adaptability to evolving urban conditions can be highly beneficial, helping MoDTs remain effective and responsive over time [11, 12].

The identified challenges should be addressed through a structured development approach that streamlines the design and implementation of MoDTs. Such an approach can reduce development costs, minimize errors, and maximize the benefits of MoDTs, ultimately promoting wider adoption of the technology. Toward this end, in this paper we present three main contributions:

- We introduce a **Model-Driven Engineering** (MDE) approach that establishes a systematic framework for MoDT development. Our framework utilizes models to organize software design across various levels of abstraction and employs automation to transform these models from high-level specifications into executable code artifacts.
- We apply the proposed approach for the development of the *MoDT for the Italian city of Bologna*. The **M2DT** (**M**odel-driven **M**obility **D**igital **T**win tool) has been developed for supporting the entire process from analysis to operationalization. This contribution validates the feasibility and effectiveness of the proposed approach in addressing the identified MoDT challenges.
- We deploy a fully operational MoDT, namely the **BoMoDT** (**Bo**logna **Mo**bility **D**igital **T**win) platform, built entirely on open-source data and technologies. The simulation accuracy and monitoring responsiveness are evaluated to assess both the platform's fidelity and effectiveness in fulfilling functional objectives.

By promoting abstraction, automation, and reusability, MDE helps align high-level system goals with low-level implementations, reducing errors, improving maintainability, and enabling adaptability. These strengths make MDE particularly suitable for managing the complex and dynamic nature of urban environments. While model-driven approaches have been explored in DTs for other domains (e.g., [13, 14]), to the best of our knowledge, this is the first work to adopt such an approach for MoDT life-cycle, covering phases from specification to deployment and operationalization, and laying the groundwork for future testing and maintenance activities.

The remainder of this paper is organized as follows. Section 2 provides an overview of MDE and model transformations. Section 3 presents the motivation and reviews related work. Section 4 introduces the proposed model-driven approach. Section 5 details its application in developing the Bologna MoDT. Section 6 outlines the deployment and quantitative evaluation of the BoMoDT platform. Section 7 assesses how the proposed approach addresses the identified MoDT challenges. Section 8 presents the conclusions and outlines potential directions for future research. Finally, Section 9 details the available code and data, including selected paper analysis, and the open-source GitHub repositories of M2DT and BoMoDT.

2. Background on Model-Driven Engineering

Model-Driven Engineering is a software development paradigm where domain-specific models act as primary artifacts throughout the system life-cycle [15, 16]. These models abstract domain knowledge and system functionality, allowing stakeholders to reason at a higher level rather than with low-level technical details [15]. Created collaboratively by product managers, designers, developers, and domain experts, such models enable automated system generation, validation, and configuration [15, 17].

To formally define modeling languages and their structure, MDE relies on the *Meta-Object Facility* (MOF) standard, introduced by the Object Management Group (OMG) [18]. MOF is based on a four-layer architecture: at the highest level, the M3 layer defines meta-metamodels, which include the MOF itself. The M2 layer defines metamodels such as the Unified Modeling Language (UML), specifying the constructs and rules of modeling languages. The M1 layer consists of models created using these metamodels, such as UML class diagrams. At the base, the M0 layer represents real-world instances described by those models. This hierarchy ensures consistency and tool interoperability. Recent advancements in MDE field brought to *multi-level modeling*, which allows elements to act simultaneously as types and instances across abstraction layers, enhancing reuse and flexibility, valuable in complex domains like DTs [19, 20].

Apart from model definitions, MDE is based on the **model transformation**, i.e., the mechanisms that systematically maps elements between models to support refinement and evolution [21]. These transformations maintain traceability and consistency, and fall into two main types: *model-to-model* (M2M) transformations, which shift between abstraction layers or views; and *model-to-text* (M2T) transformations, which generate code or configuration from models [15]. Transformations may be specified in declarative languages (e.g., ATL, QVT) or general-purpose languages.

Among the various methodologies within the MDE paradigm, this work adopts the **Model-Driven Architecture** (MDA) framework, standardized by OMG [22], due to its clear

[1] https://en.wikipedia.org/wiki/Virtual_Singapore
[2] https://smartdublin.ie/
[3] https://helsinkismart.fi/
[4] https://www.vu.city/cities/new-york

separation of concerns, support for abstraction, and strong industrial recognition [15, 17, 23]. MDA structures development across four layers: the *Computation-Independent Model* (CIM) defines business context and requirements; the *Platform-Independent Model* (PIM) specifies system functionality independent of technology; the *Platform-Specific Model* (PSM) refines the PIM with platform details; and the *Implementation-Specific Model* (ISM) defines deployable artifacts. This organization enhances traceability, reuse, maintainability, and scalability throughout the development process.

3. Motivation and Related Work

In this section, we first describe the key challenges that act as major barriers to the design and development of MoDTs, identified through a literature analysis. We then review some related work on the adoption of model-driven approaches for DT development to highlight their key contributions and limitations.

3.1. Motivation

Although MoDTs hold the potential to revolutionize urban planning, traffic management, and policy-making by providing data-driven insights to tackle congestion, sustainability, and transportation resilience, their design and development face significant challenges that hinder widespread adoption. These challenges stem from various technical and operational barriers [2, 4]. Table 1 outlines some of the key *challenges* reported in the related literature that we identified by means of an opportunistic literature review. While not exhaustive, the challenges' list is grounded in the selected studies and reflects the main barriers currently affecting the development and adoption of MoDTs.

To conduct the review [34], on January 2025 we performed a search on IEEE Xplore[5], ACM Digital Library[6], and Scopus[7] using the query[8]: "*Digital Twin*" AND ("*Urban Mobility*" OR "*Urban Transportation*"). Since a detailed literature review is beyond the scope of this paper, we provide only a brief overview of the process. The initial search yielded 444 manuscripts, published between 2017 and 2024, comprising peer-reviewed journal articles and conference proceedings.

Manuscripts were excluded if they met any of the following criteria: *(i)* not related to urban mobility or transportation; *(ii)* not focused on DTs architecting or engineering; *(iii)* not related to DTs, but addressed general simulation; or *(iv)* were not primary studies, or were secondary studies lacking a scientific proposal. Only manuscripts that met none of these exclusion criteria were selected, resulting in a final set of 11 papers.. To reduce selection bias and capture additional relevant work [35], we applied backward and forward snowballing techniques [36], which led to the inclusion of 7 more papers, bringing the total to *18 selected studies*. Readers interested in the search, selection and manuscripts' analysis can refer to the replication package available at: https://docs.google.com/spreadsheets/d/1hNPAHboFAdLjr3NsHmt4lbSlIULCiWyF/edit?usp=sharing&ouid=110130922453565990010&rtpof=true&sd=true.

C_1—**Composability** refers to the structured integration of lower-level DTs, which represent specific mobility components, into higher-level DTs that represent the broader city system [24]. As illustrated in Figure 1, a MoDT is designed as a *Digital Twin Composite* [26, 37]: lower-level DTs, such as those modeling traffic flow, public transportation, and ride-sharing services, are seamlessly integrated into a higher-level DT, i.e., the MoDT. Furthermore, the concept of composition can be extended to the entire urban landscape, where a MoDT, which is the mobility-centered DT, combined with twins of energy, weather, and other urban aspects, forms an *Urban Digital Twin* (UDT). The UDT provides a comprehensive digital replica of the (smart) city, i.e., the physical city augmented by digital technologies and services that the DT seeks to monitor and control [27].

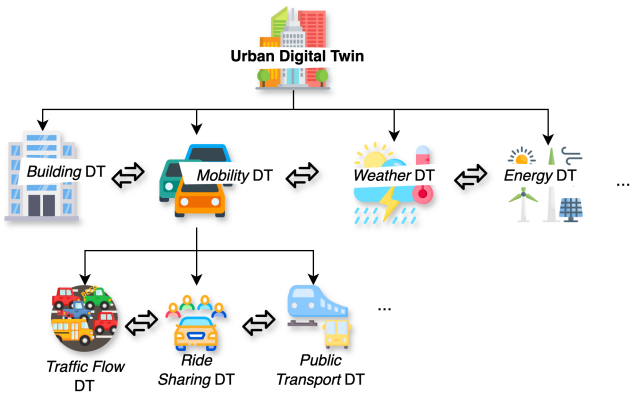


Figure 1: Mobility Digital Twins structure.

C_2—**Interoperability** enables MoDTs to seamlessly interpret, integrate, and exchange heterogeneous mobility data across diverse infrastructures while ensuring consistency and clarity. Mobility data come from multiple sources, such as traffic monitoring sensors, GPS tracking systems, public transportation networks, surveillance cameras, urban planning models. As a MoDT is composed of multiple application-specific DTs, these constituent DTs often act as data providers, for example ride-sharing companies can supply vehicle usage data. Similarly, the UDT may offer inputs, such as weather conditions or infrastructure status, which can be essential for AI-driven services within the MoDT. Conversely, the MoDT can supply real-time mobility insights back to the UDT. This dynamic exchange of data across multiple systems and providers highlights the need for robust interoperability [38], not only at the data level but also across services, interfaces, and models. Achieving the composite structure of MoDTs (and UDTs) therefore demands harmonization among independently developed components [9, 28, 30].

[5] https://ieeexplore.ieee.org/
[6] https://dl.acm.org/
[7] https://www.scopus.com/
[8] Google Scholar was excluded to ensure the inclusion of only high-quality, peer-reviewed sources. Web of Science was omitted due to its broad interdisciplinary scope, which may reduce the focus required for a domain-centered review.

Table 1: Key Challenges of Mobility Digital Twins identified from literature.

ID	Name	Description	Refs
C_1	Composability	The ability of a MoDT to integrate lower-level DTs within a unified system.	[1, 2, 4, 5, 7, 9, 24–26]
C_2	Interoperability	The capacity of MoDT to accurately interpret, integrate, and exchange heterogeneous mobility data across various infrastructures while maintaining consistency and clarity.	[5, 7, 9, 24, 27–31]
C_3	Scalability	The ability of MoDT to efficiently manage increasing volumes of data and demands, while ensuring computational efficiency, responsiveness, and interoperability.	[1, 2, 4, 5, 7, 24, 32]
C_4	Adaptability	The capability of MoDT to dynamically adjust to evolving infrastructure, sensor networks, and traffic conditions while preserving operational stability.	[2, 9, 11, 12, 28, 31, 33]
C_5	Maintainability	The ease with which MoDT can be modified, updated, and extended to accommodate changes without compromising functionality or structural integrity.	[4, 11, 24, 29]
C_6	Automatability	The degree to which MoDT development processes can be automated to improve efficiency, minimize errors, and enhance system evolution over time.	[4, 11, 12, 28]

C_3**—Scalability** refers to the ability of MoDTs to manage increasing demands in terms of real-world data volume, computational load, service interactions, while preserving computational efficiency, responsiveness, and seamless integration. Unlike DTs in controlled settings like manufacturing, where conditions remain stable and predictable, MoDTs function within dynamic, heterogeneous, and continuously evolving urban environments. As they grow in scale, they must handle huge incoming data, user interactions, simulation workloads, and coordination between distributed components. Ensuring scalability is therefore essential to accommodate rising data flows and service demands while maintaining performance and interoperability in ever-changing urban landscapes [1, 4].

C_4**—Adaptability** is MoDTs' capability to dynamically respond to changes in infrastructure, sensor networks, and traffic conditions while maintaining operational stability. To remain effective, MoDTs must quickly adapt to external shifts, often within short cycles, without requiring extensive re-engineering or disrupting ongoing operations. As urban environments evolve, MoDTs may need to incorporate new sensors, adjust parameters, or update models to accurately reflect real-world conditions, making adaptability a crucial aspect of their development to ensure continuous alignment with shifting urban dynamics [11, 12].

C_5**—Maintainability** represents the ease with which MoDTs can be modified, updated, and extended while preserving functionality and structural integrity. Given their complexity, long-term sustainability depends on efficient modification and debugging processes to prevent system degradation. Ensuring maintainability allows MoDTs to evolve seamlessly while preventing the accumulation of technical debt, which can otherwise lead to increased operational costs and inefficiencies [28]. Well-defined maintainability strategies enhance the reliability, scalability, and adaptability of MoDTs over time, minimizing disruptions and optimizing overall system performance [4].

C_6**—Automatability** refers to the extent to which MoDT development processes can be automated to enhance efficiency, reduce errors, and support continuous system evolution. Given the dynamic nature of urban environments, automatability plays a critical role in streamlining engineering workflows, minimizing manual intervention, and improving overall accuracy throughout the development pipeline [11, 12]. Automation also reinforces adaptability to changing requirements while preserving structural integrity, system stability, and long-term maintainability. Furthermore, automated workflows improve traceability, allowing seamless propagation of modifications across interconnected components. By embedding automation into MoDT engineering, development cycles become more efficient, scalable, and resilient, aligning with the increasing complexity of urban mobility systems.

While some of these challenges can be shared with DTs of other systems, they manifest in unique ways in the context of MoDTs, such as the integration of vast and heterogeneous urban data sources, and coordination across diverse spatial and organizational scales [5]. These domain-specific factors significantly increase both the technical and operational complexity of MoDT development, as previously discussed, and continue to impede their adoption. Consequently, the full potential of MoDTs to improve urban transportation systems, and more broadly to support the creation of smarter and more resilient cities, remains largely untapped. Addressing these challenges requires a comprehensive, end-to-end approach throughout the MoDT life-cycle, aimed at fostering innovation, simplifying development, and ultimately enabling more intelligent and adaptable urban mobility solutions.

3.2. Related Work

This work addresses the challenges C_1 to C_6 in the design and development of MoDTs through a structured and systematic *model-driven* approach. To demonstrate its applicability, the proposed approach is adopted in the development of the Bologna MoDT, validating its feasibility and effectiveness in tackling the identified challenges.

Despite the existence of standardized DT architecture in manufacturing [39, 40] and even though previous studies have explored the DTs engineering and architecting through model-driven methodologies across various domains, they primarily focus on specific aspects of twin development, such as modeling and simulation [41, 42], human interaction and interfacing with DTs [13, 43, 44], scenario evolution in industrial applications [14, 45] and code generation [46–48]. For example, Muñoz *et al.* [43] proposed an MDE framework that integrates UML and Object Constraint Language to define behavioral models while leveraging a data lake for bidirectional interaction between physical and digital components. Similarly, Lehner *et al.* [45] employed fluent APIs within an MDE approach to manage DT evolution alongside changes in physical

systems. However, by concentrating on some phases of DT development, these studies lack a holistic, end-to-end perspective, leading to solutions that struggle in addressing challenges and complexities that arise throughout the entire DT life-cycle.

Furthermore, many existing solutions rely on proprietary or highly customized DT development frameworks. For example, Michael *et al.* [47] applied MDE with the MontiGem code generation framework to develop low-code DT platforms. Similarly, Dalibor *et al.* [44] employed an MDA approach to develop DT interfaces, utilizing MontiArc and MontiGem for automated code generation. However, these studies do not explore the integration of existing DT platforms such as Eclipse Ditto[9], Azure Digital Twins[10], and FIWARE[11] [49], resulting in use case specific solutions, which hinder their reusability, adaptability and modifiability, while increasing development complexities and costs.

To the best of our knowledge, none of the existing approaches offers a comprehensive model-driven solution that spans DT development from initial specification to full deployment and operationalization. Moreover, rather than introducing yet another specialized DT framework, our approach builds upon and integrates existing technologies, leveraging their strengths to create a more adaptable and reusable solution. In particular, we chose *FIWARE*, a widely adopted open-source platform for DT development, and *Eclipse SUMO*, a well-established urban traffic simulator, as the target technologies for the implementation of our case study. To summarize, our work complements existing research by providing a concrete example of how a model-driven approach can be used to effectively enhance MoDT development. Moreover, as it will be discussed in Sec. 7, the proposed approach helps address the previously identified challenges, thus strengthening the overall effectiveness of the obtained MoDT solution.

4. Model-Driven Approach for Developing MoDTs

This section introduces the proposed model-driven approach for designing and developing MoDTs. As anticipated in Sec. 2, we adopt a *Model-Driven Architecture* framework organized in four modeling layers. Each level plays a distinct role, guiding the progression from high-level context and requirements to abstract system design, through platform-aware refinement, and ultimately to the generation of executable code artifacts. Our goal is to adopt an approach that easily supports the MoDT development: like other DTs, a MoDT is a software system, and its life-cycle follows classical software and systems engineering phases, adapted to the urban mobility domain. Therefore, it begins with the analysis of domain-specific MoDT requirements and available data sources, followed by the architectural design. MoDT development involves implementing this design using suitable technologies and programming languages. After deployment and integration into the operational environment, testing and validation ensure correct system behavior, while maintenance supports continuous updates in response to evolving policies, infrastructure, or data.

By enabling a systematic refinement from abstract requirements to executable code through model transformations, MDA provides methodological support for all standard phases of software development and thus is well-suited for MoDT development, facilitating traceability across phases, supporting testing and simplifying maintenance [15]. Figure 2 illustrates the activity diagram which maps the four modeling levels to the different steps with required inputs and models produced at each stage. Each modeling level is detailed in the following paragraphs.

Computational Independent Level. This is the highest level of abstraction, defining the system context, requirements, and objectives without considering computational constraints, resulting in the CIM model [15]. As illustrated in Fig. 2, our approach employs a dual-domain analysis to clearly distinguish between the twin system and its application environment.

The *Digital Twin analysis* aims to define the core structure and functionalities of the Digital Twin, independently of any specific application domain. It takes as input specifications derived from sources such as informal descriptions, requirement documents, user stories, and use case diagrams. These typically include domain-independent capabilities such as maintaining synchronization with physical counterparts, enabling bidirectional communication, supporting runtime monitoring and control, and ensuring data consistency across digital components [50, 51], which represent general DT requirements applicable across domains, therefore independent of mobility or any specific application area.

The output of this step is the Digital Twin model, a conceptual representation that formalizes the key elements of a DT and their relations. These elements may include components responsible for managing bidirectional synchronization between the physical and digital spaces, as well as elements supporting modeling and simulation. The interactions are captured through defined relationships among these elements, such as data flows, control dependencies, or coordination mechanisms.

In parallel, the *mobility domain analysis* is carried out to define the structure of the urban mobility system that the Digital Twin will replicate. Input specifications may include requirements that define aspects such as strategies for sensor deployment or the implementation of traffic control rules. User stories can reflect stakeholder needs, such as a traffic engineer evaluating congestion during peak hours or a city planner examining the impact of different traffic signal configurations [1, 2]. These inputs are analyzed for designing the mobility models, which represents the physical infrastructure of the real-world urban environment.

By distinguishing these two analyses, the proposed approach ensures a clear separation between domain-expert knowledge provided by mobility specialists, who are typically well-versed in the operational domain but not in DT systems, and the technical expertise of DT specialists, who focus on the overall DT system requirements. This structured differentiation allows each aspect to be specified independently while laying the ground-

[9] https://eclipse.dev/ditto/

[10] https://azure.microsoft.com/en-us/products/digital-twins/

[11] http://fiware.org/

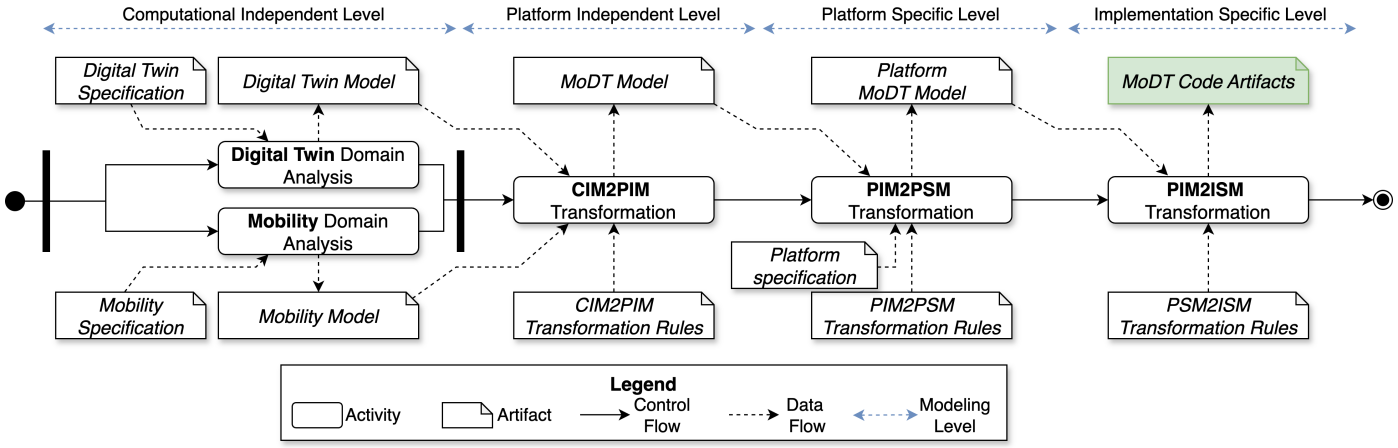


Figure 2: Model-Driven Architecture for MoDTs.

work for their seamless integration in subsequent modeling phase.

Platform Independent Level. This level defines the structure of the application without being tied to a specific implementation platform, resulting in a Platform Independent Model, which maintains a sufficient level of abstraction to allow mapping to one or more concrete implementation platforms [15]. As shown in Fig. 2, this level involves a *model-to-model transformation* activity that takes as input the DT model, the mobility model, and the set of transformation rules that guide their integration.

The resulting MoDT model unifies the Digital Twin system definition and the mobility system description, providing a comprehensive representation of the MoDT structure while maintaining platform independence. For instance, the MoDT model may define which elements of the physical environment are digitally represented, such as roads, intersections, traffic signals, and vehicles, as well as the DT services provided, including traffic state monitoring, vehicle tracking, mobility data visualization, and simulation-based analysis of vehicle movements [1, 4].

Platform Specific Level. This level defines a model that incorporates all necessary details about the application's structure on a specific platform, providing developers with the information needed to generate executable code [15]. As depicted in Fig. 2, this stage involves another *model-to-model transformation*, which takes as input the MoDT model, a set of transformation rules, and the platform specifications. The output is the platform specific MoDT model, which tailors the mobility DT system to the selected platform(s), ensuring alignment with the targeted infrastructure and operational environment.

For example, the PSM model can introduce concrete technical details of a specific traffic simulator, such as Eclipse SUMO, PTV Vissim, or MATSim, or the visualization frameworks, such as CesiumJS for 3D city mapping. Additionally, the model may define how vehicle telemetry data are ingested using MQTT brokers, how traffic events are processed through a stream processing engine in compliance with high-level requirements, and how data flows are managed across the platform components.

Implementation Specific Level. The final stage of the approach focuses on generating the Implementation Specific Model, which consists of code artifacts written in a programming language to produce executable software. This step involves a *model-to-text transformation*, which takes as input the platform specific MoDT model along with transformation rules adapted to the selected programming language. The resulting MoDT code artifacts can undergo further refinement before being deployed.

5. The Bologna MoDT Case Study

This section describes the application of the proposed approach for the development of Bologna MoDT. In particular, Sec. 5.1 provides an overview of how we carried out the different steps of the MDA approach by distinguishing among manual and automated activities, performed by means of the **M**odel-driven **M**obility **D**igital **T**win (**M2DT**) **tool**, developed ad-hoc for enabling model transformations devised by the approach. Sections 5.2 to 5.5 discuss in detail each modeling level and the corresponding rules defined for automated transformations, progressing from high-level models to concrete implementation artifacts. Throughout the steps, UML class diagrams were chosen to represent both input and output models. These diagrams focus on structural elements and inter-class relationships, and do not include attributes or methods, in order to maintain conceptual clarity and reduce visual complexity.

5.1. Implementing Bologna MoDT

As anticipated, the model-driven approach for MoDTs, illustrated in Fig. 2, was applied to the Bologna case study through a combination of manual and automated activities. At the **CIM level**, the devised *domain analyses*, which result in the design of the DT CIM and Bologna mobility models, were manually carried out due to the nature of their respective data sources. The DT domain analysis draws upon our previous research work on DT architectures, as detailed in Sec. 5.2. In contrast, Bologna model is based on open-source mobility data, which required

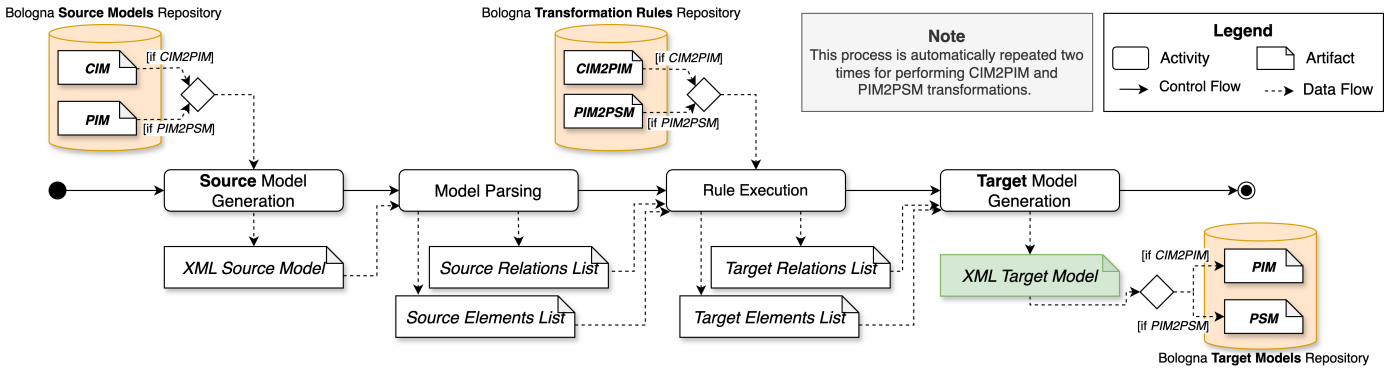


Figure 3: M2DT tool.

manual interpretation and structuring to accurately capture the city mobility context.

At the **PIM** and **PSM levels**, *two model-to-model transformations*, refining the CIM into a platform independent model (CIM2PIM), and transforming the PIM into a platform specific model (PIM2PSM), were automatically executed using our **M2DT tool** available at: `https://github.com/alessandrasomma28/m2dt`. We developed the M2DT tool in Python to enhance usability and maintainability, particularly for developers and DT practitioners who may not have expertise in specialized model transformation languages. This choice ensures that M2DT remains lightweight and easily extensible, enabling rapid customization of transformation rules and straightforward integration with deployment and simulation workflows. By leveraging Python's extensive ecosystem of libraries, the tool simplifies the transformation process and better supports rapid prototyping and iterative development in real-world scenarios.

Figure 3 illustrates the sequence of activities involved in M2DT. Since both transformations follow the same workflow, the figure presents the steps only once to avoid redundancy, while highlighting the different inputs for each case. The M2DT tool executes four main steps:

1. *Source Model Generation*: The source model is converted into an eXtensible Markup Language (XML)-based representation. Depending on the transformation, the source model can be either the CIM or the PIM.
2. *Model Parsing*: From the XML representation, the tool extracts all elements and relationships, organizing them into structured lists that represent the source model's components.
3. *Rule Execution*: Based on the transformation type, the appropriate set of rules is applied to convert the source elements and relationships into target elements and relationships.
4. *Target Model Generation*: Finally, the transformed elements and relations are assembled into a new XML file representing the target model. This model, which can be the PIM or the PSM, can be visualized in any tool that supports XML-based visualization of UML class diagrams.

As explained, the tool operates by taking as input a source model along with a set of transformation rules, implemented as Python functions, and produces the corresponding target model as output. While this implementation is based on Python for simplicity and extensibility, it does not constrain the general applicability of the approach. On the contrary, developers can either reuse the M2DT tool directly by providing new source models, or alternatively, they can adapt the tool by re-implementing the transformation rules in other languages, such as ATL or general-purpose programming languages like Java, depending on the specific requirements of the use case.

Finally, at the **ISM level**, the *Model-to-Text transformation* activity was automatically executed to generate implementation-specific artifacts from the platform specific model. This PSM2ISM transformation is carried out using the code generation tool integrated in Visual Paradigm[12], the modeling software used in our case study to visualize UML diagrams.

5.2. Computational Independent Level

This section presents the models developed at the computation independent level, derived from two separate domain analyses. While these activities are described in sequence, they are completely independent processes. Each begins with distinct specifications and can be carried out in parallel by domain experts within their respective areas of knowledge.

DT domain analysis. This activity builds on our earlier work on designing DT architectures [49], in which we performed a systematic review of state-of-the-art DT solutions to identify the most frequently used architectural elements and their interconnections. These findings guided the design of TwinArch, a Digital Twin Reference Architecture, organized into multiple architectural views[13]. The architecture was validated through an online survey with DTs experts from both academia and industry.

Based on this structured design and validation process, we selected one specific TwinArch view, modeling DT domain entities using an UML class diagram, as the foundation for our approach. The extracted diagram defines the **Digital Twin CIM**

[12] `https://www.visual-paradigm.com/`

[13] An architectural view refers to specific subsets of system elements and their relationships, tailored to address the concerns of particular stakeholders [52].

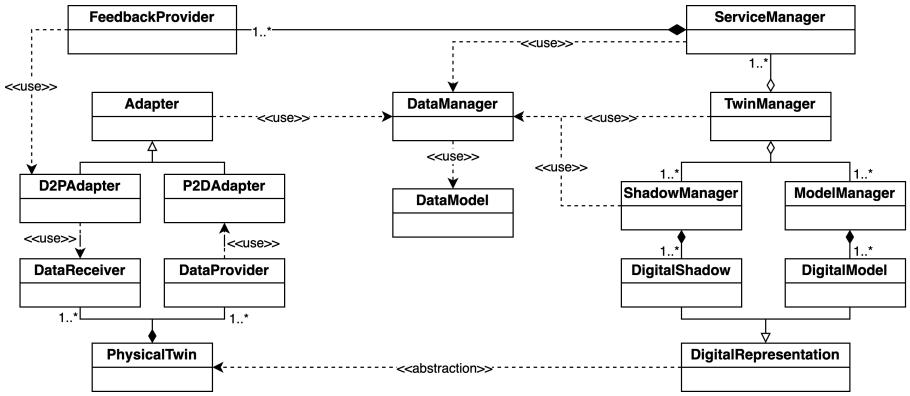


Figure 4: Digital Twin CIM from [49].

and is shown in Figure 4. At its core is the distinction between the PhysicalTwin, representing the real-world system, and its virtual counterpart, the DigitalRepresentation. The digital representation is specialized into two main types. The DigitalShadow captures temporal data reflecting the physical system state over time, enabling anomaly detection, historical analysis, and state monitoring. The DigitalModel encodes behavioral logic, supporting simulation, prediction, and scenario analysis. Multiple instances of each can coexist and are managed by the ShadowManager and the ModelManager, respectively. Their coordination is handled by the TwinManager, which aligns data-driven and behavior-driven components.

The ServiceManager handles high-level operations such as monitoring, prediction, and analysis. When one or more DT-based services require interaction with the physical system, the service manager communicates with the FeedbackProvider, which generates alerts, events, and control commands directed towards the physical environment. Bidirectional communication is managed by the DataProvider and the DataReceiver, responsible for collecting data from the physical world and delivering feedback to it, respectively. These are supported by the two adapters, the P2DAdapter which handles physical-to-digital data flow, and the D2PAdapter that manages digital-to-physical communication. The number of adapters depends on the specific requirements of the real world context. For example, an adapter may be defined for each category of data provider, or multiple adapters may be required for providers of the same type, such as sensors from different manufacturers that expose distinct interface protocols. Finally, the DataManager oversees data aggregation, storage, and retrieval, supported by the DataModel, which defines the structure of exchanged data and ensures interoperability.

Mobility domain analysis of Bologna. This step was manually carried out using available mobility-related data for Bologna, a mid-size city in Italy's Emilia-Romagna region. Bologna has implemented various initiatives to enhance urban mobility and sustainability, such as the *Città 30* project[14], which enforces a city-wide speed limit of 30 km/h, and the Bologna Open Data platform[15], which provides access to traffic-related datasets.

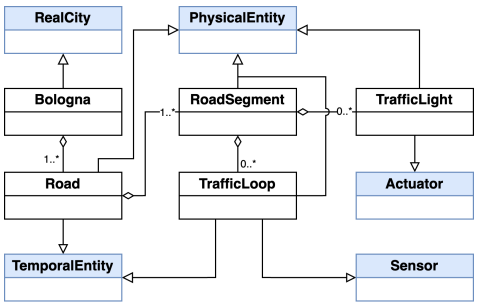


Figure 5: Bologna Mobility CIM: classes marked in blue represent functional abstraction of domain-related entities.

The primary dataset used in this analysis includes traffic flow measurements collected by a network of induction loop detectors, which have been operational since 2019 and are distributed throughout Bologna's road network. Each record contains temporal information (such as date, time, and day of the week), spatial details (including road name, traffic direction, and geographic coordinates), and vehicle count data. Since information about the actuation components of the traffic infrastructure (e.g., traffic lights) is not publicly available, we relied on the dataset provided in [53]. This dataset, released by Bologna municipality for simulation purposes, includes the geographic locations and control logic of actual traffic lights.

Based on these data sources, we identified the key entities involved in traffic monitoring and control within the city. These were then abstracted into a computation independent model, represented as UML class diagram illustrated in Figure 5. White classes represent domain-specific elements directly derived from the data, while blue classes are meta-classes that describe their abstract functional roles within the mobility context. Class named Bologna models the real city and is composed of multiple Road instances, each subdivided into one or more RoadSegment elements. Traffic data are collected through 312 TrafficLoop sensors installed on road segments, while traffic flow is managed by 140 TrafficLight actuators located at intersections.

[14] https://www.bolognacitta30.it/

[15] https://opendata.comune.bologna.it

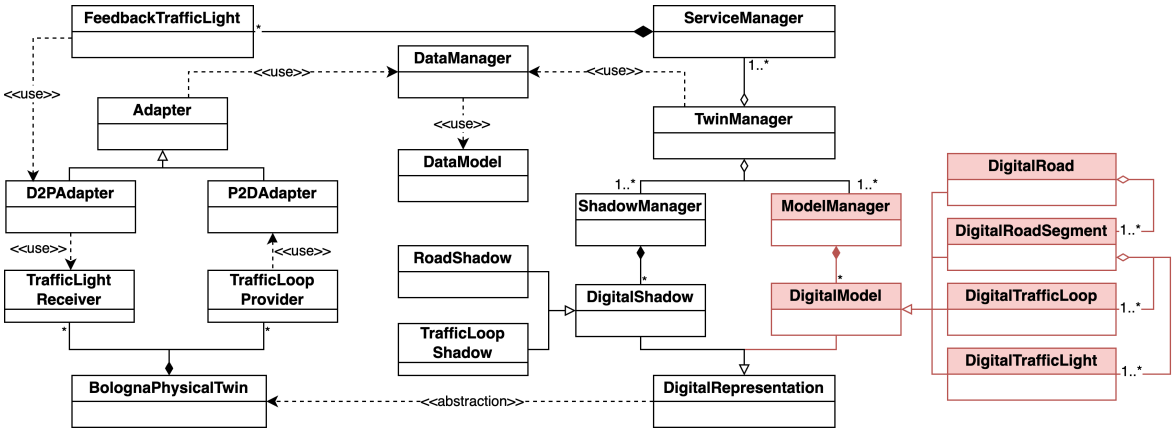


Figure 6: Bologna MoDT PIM: red classes and relations represent rule CPR_2 outcome.

Each domain element is linked to meta-classes that reflect its functional abstraction. Bologna is associated with RealCity, representing the physical urban environment being modeled. The infrastructural components (roads, segments, traffic loops, and traffic lights) are all categorized as PhysicalEntity, as they represent tangible elements of the urban space. Additionally, TrafficLoop plays the role of a Sensor, since it captures traffic data, and a TemporalEntity, given its time-dependent measurements. TrafficLight is also modeled as an Actuator, as it controls vehicle movement by enforcing signal logic.

5.3. Platform Independent Level

This section describes the Bologna MoDT model, which results from the CIM2PIM transformation performed at the platform independent level. This activity is carried out by the M2DT tool, which takes as input the DT and Bologna CIM models, along with the set of CIM2PIM transformation rules, and produces the target PIM model. To ensure clarity, we first present the output of the transformation, i.e., the Bologna MoDT PIM, before introducing the transformation rules we defined. This structure avoids confusion and allows readers to understand the PIM entities before encountering the rules that produce them, making the role and purpose of each rule easier to follow.

Figure 6 shows the resulting **Bologna MoDT PIM**, where the BolognaPhysicalTwin class represents the real-world Bologna city, abstracted by the DigitalRepresentation serving as the virtual counterpart. In line with the already discussed DT model, the digital representation is composed of two complementary entities: shadows and models. The shadow instances, such as RoadShadow and TrafficLoopShadow, capture time-dependent traffic data relevant to the Bologna context. Meanwhile, the digital model is specialized into digital counterparts of physical entities identified in the Bologna CIM, namely, DigitalRoad, DigitalRoadSegment, DigitalTrafficLoop, and DigitalTrafficLight. These represent the structural and behavioral aspects of the urban infrastructure. The coordination of the digital representations is managed by the shadow and model managers, with overall orchestration handled by the TwinManager, ensuring integration between data-driven and model-driven perspectives.

Bidirectional communication between the physical and digital spaces is implemented based on the sensor and actuator roles defined in the Bologna CIM. The TrafficLoopProvider captures traffic data from induction loops, while the TrafficLightReceiver delivers control commands to traffic lights. These interfaces are connected through the adapters, which ensure seamless data exchange in both directions. Data consistency and storage are managed by the DataManager, in conjunction with the DataModel. Finally, high-level MoDT services, such as monitoring and control, are handled by the ServiceManager, while feedback mechanisms are encapsulated in the FeedbackTrafficLight class, enabling the system to respond to dynamic traffic conditions.

CIM2PIM Transformation Rules. To automate the transformation of computation independent entities and relations into the Bologna MoDT model described above, eight CIM2PIM transformation rules have been defined. General rules are defined in Table 2, which reports their identifiers, names, objectives, required CIM input entities, and corresponding PIM output entities. The application of these rules to the Bologna case study is detailed in Table 3, which specifies the actual CIM entities provided as input and the resulting MoDT entities produced as output. It is important to note that although both tables list only the involved classes, the transformation rules and their implementation in the M2DT tool also account for the relationships among entities, including compositions, aggregations, and usage dependencies.

Rule CPR_1, named mapToPhysicalTwin, is designed to map each real-world city entity to its corresponding physical twin class in the PIM. This step is essential to ensure that the final MoDT system includes its physical counterpart. Without the physical entity, the foundation of a MoDT would be missing, making it impossible to create a dynamic and evolving digital representation of the real urban environment. Applying CPR_1 to Bologna use case means that Bologna class is transformed in the BolognaPhysicalTwin class.

Rule CPR_2, named digitalizePhysicalEntity, focuses on converting all physical entities to be virtually replicated into their digital counterparts in the PIM. This means that traffic infrastructure classes, such as Road and RoadSegment, are transformed into their digital equivalent, e.g. DigitalRoad. The out-

Table 2: CIM2PIM Transformation Rules.

ID	Name	Description	Input CIM Entities	Output PIM Entities
CPR_1	*mapToPhysicalTwin*	Maps real-world entities to their corresponding physical twin representation.	City Environment	PhysicalTwin
CPR_2	*digitalizePhysicalEntity*	Converts all physical entities into their digital counterparts.	Physical Entities	Digital Models, Model Manager
CPR_3	*transformTemporalEntity*	Transforms temporal entities into digital shadows representing time-varying state.	Temporal Entities	Digital Shadows, Shadow Manager
CPR_4	*mergeShadowModelFlow*	Integrates model and shadow information into unified representation.	—	Digital Representations, Twin Manager
CPR_5	*transformSensor*	Maps sensor roles to digital data providers and adapters.	Sensor	Data Providers, Provider Adapters
CPR_6	*transformActuator*	Maps actuator roles to digital data receivers and adapters.	Actuator	Data Receivers, Receiver Adapters
CPR_7	*integrateServiceFeedback*	Integrates service management and feedback mechanisms into the system.	Service, Actuator	Service Manager, Feedback Entities
CPR_8	*integrateDataManager*	Adds data management and data modeling components for consistent handling.	—	Data Manager, Data Model

Table 3: CIM2PIM Rules Applied to Bologna case study.

ID	Input CIM Entity	Output PIM Entity
CPR_1	Bologna, RealCity	BolognaPhysicalTwin
CPR_2	PhysicalEntity, Road, RoadSegment, TrafficLoop, TrafficLight	DigitalRoad, DigitalRoadSegment, DigitalTrafficLight, DigitalTrafficLoop, DigitalModel, ModelManager
CPR_3	TemporalEntity, TrafficLoop, Road	TrafficLoopShadow, RoadShadow, DigitalShadow, ShadowManager
CPR_4	—	DigitalRepresentation, TwinManager
CPR_5	Sensor, TrafficLoop	TrafficLoopProvider, P2DAdapter, Adapter
CPR_6	Actuator, TrafficLight	TrafficLightReceiver, D2PAdapter
CPR_7	TrafficLight, Actuator	ServiceManager, FeedbackTrafficLight
CPR_8	—	DataManager, DataModel

come of CPR_2 execution is shown with red-colored classes and relations in Fig. 6. The pseudocode for implementing CPR_2 is shown in Algorithm 1, which illustrates how the function identifies and transforms all child classes of the PhysicalEntity meta-class. Moreover, the rule also preserves the relationships

Algorithm 1 CPR_2: digitalizePhysicalEntity

Require: *cimClasses*, *cimRelations*
Ensure: Updated *pimClasses*, *pimRelations*
Identify *PhysicalEntity* in *cimClasses*
Get child classes of *PhysicalEntity*
for each c_i (child class) **do**
  p_i.name ← 'Digital' + c_i
  p_i.id ← ID
  Add p_i to *pimClasses*
end for
for each c_i **do**
  for each relation $R_i(c_i, c_j)$ in *cimClasses* **do**
    Find p_i, p_j corresponding to c_i, c_j
    Create relation $r_i(p_i, p_j)$
    Add r_i to *pimRelations*
  end for
end for
return Updated *pimClasses*, *pimRelations*

defined in the CIM, ensuring that the structure and semantics of the original physical system are maintained in the PIM, for instance the aggregation between a road and its segments.

Rule CPR_3, named transformTemporalEntity, focuses on time-sensitive entities transforming them into digital shadows, which are data traces that capture snapshots of the city's state over time [49]. For example, road shadows may represent variations in traffic flow measurements and, when applicable, the state of traffic lights associated with those roads. Executing CPR_3 on Bologna CIM results in the TrafficLoopShadow and RoadShadow and the related management entities and relations.

Rule CPR_4, named mergeShadowModelFlow, complements the CPR_2 and CPR_3 rules by integrating the information flows from both digital models and digital shadows. This integration occurs from the bottom through digital representation entity that abstracts both sources, and from the top through the TwinManager, which coordinates the two flows across the system.

Rules CPR_5 and CPR_6 address the sensing and actuation components of the system. More in detail, the transformSensor rule identifies entities that serve as sensors and maps them to their corresponding data providers in the PIM. For example, a traffic loop sensor in the Bologna CIM is transformed into a TrafficLoopProvider, enabling the flow of data from the physical city into its DT. Similarly, the transformActuator rule maps CIM entities with actuation roles to their respective data receivers in the PIM. Both rules introduce adapters that enable smooth and consistent data exchange between the physical and digital layers of the system.

Rule CPR_7, entitled integrateServiceFeedback, introduces the ServiceManager into the PIM to manage all DT-based services. It also integrates the FeedbackProvider entities responsible for generating feedback mechanisms, specialized for the type of physical entities to which alerts or commands can be sent. In the Bologna case study, these entities are primarily traffic lights actuators, resulting in the creation of the FeedbackTrafficLight entity. Finally, rule CPR_8 incorporates entities for data management and modeling, the DataManager and DataModel, to ensure consistent and structured data handling across all components of the MoDT ecosystem.

5.4. Platform Specific Level

This section presents the Bologna MoDT model tailored to the selected technologies, which is the result of the PIM2PSM transformation performed at the platform specific level. This transformation is carried out using the M2DT tool, which takes as input the PIM model, the predefined set of PIM2PSM transformation rules, and the technology specifications, and produces the target PSM model. As discussed in Section 3.2, Eclipse SUMO was chosen for traffic simulation and FIWARE for data management. These technologies were selected to

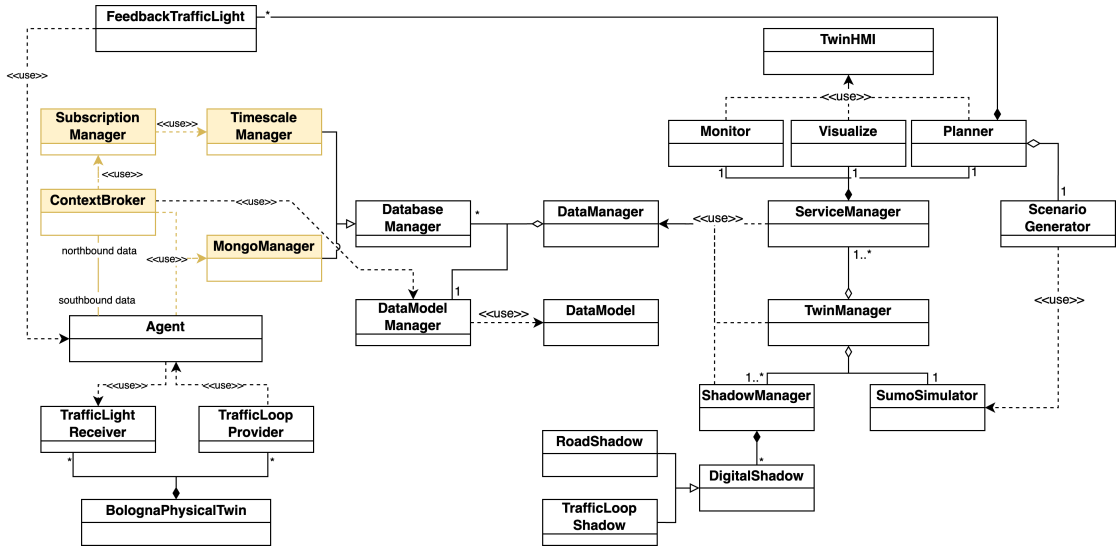


Figure 7: Bologna MoDT PSM: yellow classes and relations represent rule PPR_2 outcome.

avoid ad-hoc implementations and to enhance interoperability, reusability, and adaptability by leveraging reliable, open-source, well-documented, and freely accessible platforms.

For completeness, we begin with a brief introduction of the two selected platforms as they influence the final PSM structure. Following the same presentation strategy used for the previous modeling level, we first describe the result of the PIM2PSM transformation, i.e., the Bologna MoDT PSM, and then we present the transformation rules we used to automate this step.

Brief Technologies Specification. *Eclipse SUMO* is an open-source, microscopic traffic simulator developed by the German Aerospace Center [54]. It allows for detailed modeling of individual road users, such as cars, buses, and pedestrians, and supports in-depth analysis of traffic conditions, including congestion and emissions. SUMO simulations are based on three main inputs: the road network, additional traffic infrastructure (such as traffic lights and induction loops), and traffic demand [53]. It provides real-time visualization through a graphical interface and produces fine-grained outputs, making it particularly suitable for integration within MoDT workflows.

FIWARE is a modular, open-source ecosystem supported by the European Commission for building smart applications, including DTs. It provides a collection of reusable software components for data sharing and management [55]. At its core is the Context Broker, which manages context data, namely the current states of physical and digital entities, in a format compliant with the Next Generation Service Interface (NGSI) standard. To ensure semantic consistency and interoperability, FIWARE promotes the use of standardized Smart Data Models (SDMs), which are defined as JSON schemas and offer consistent data structures for use across different application domains [30]. In our case study, we adopted SDMs for the Transportation domain [31, 56]. Additional FIWARE components include the IoT Agent, which enables adaptation between device-specific communication protocols and the NGSI format, and QuantumLeap, which addresses the Context Broker's limitation of only storing the latest entity state by providing support for time-series data storage.

Figure 7 shows the resulting **Bologna MoDT PSM**. The physical twin, its data providers (traffic loops), and receivers (traffic lights) remain unchanged, as do the digital shadow elements and their manager. This is because the PIM2PSM transformation only alters components affected by the chosen technologies, leaving the physical setup and digital shadow logic intact. In contrast, digital modeling and management elements are merged into a new class, SumoSimulator, reflecting Eclipse SUMO's full support for traffic modeling and simulation. The TwinManager entity is maintained to orchestrate the interaction between the simulator and the digital shadows, ensuring that simulations are run with appropriate inputs derived from real-world data.

For service-related aspects, the ServiceManager has been decomposed into three distinct classes to handle considered MoDT functions: traffic scenario planning and simulation, data visualization, and monitoring of the real-world urban system. Predictive capabilities are not included in this work, and thus entities responsible for tasks such as AI-based prediction are not modeled. More in detail, the newly added classes are:

- Planner defines traffic scenarios based on real-world conditions and planning goals. These are passed to the ScenarioGenerator, which produces SUMO-compatible inputs for simulation. After analyzing the simulation results, the planner can trigger alerts or send control commands to the physical system through the FeedbackTrafficLight entity.
- Visualizer class displays both real and simulated data, providing users with a clear, user-friendly view of system behavior.
- Monitor tracks the state of the physical system, enabling users to maintain situational awareness and understand real city's current conditions.

All services interact through the TwinHMI, which acts as the main interface for viewing the physical system, running sim-

Table 4: PIM2PSM Transformation Rules.

ID	Name	Description	Input PIM Entities	Output PSM Entities
PPR_1	*transformDigitalModel*	Maps digital model components into a unified simulation component.	DigitalRepresentation, DigitalModel, ModelManager, Digital Entities	Simulator
PPR_2	*createFiwareContext*	Introduces FIWARE components for context data handling.	—	Context Broker, Historical Broker, Storage Manager
PPR_3	*transformAdapter*	Transforms adapters and interface elements into FIWARE-compatible agents.	Adapters, Sensors, Actuators	IoT Agents
PPR_4	*transformService*	Refines services into concrete elements for specific services and user interaction.	Service and Twin Manager, Feedback Entities	User Interface, Services Entities
PPR_5	*integrateData*	Adds components for data modeling and storage across multiple platforms.	Data Manager and Modeling	Data Storage and Management Entities
PPR_6	*updateShadowRelation*	Preserves and maps temporal entities not affected by technological choice.	Digital Shadows	Digital Shadows and Manager

Table 5: PIM2PSM Rules Applied to Bologna case study.

ID	Input PIM Entity	Output PSM Entity
PPR_1	DigitalRepresentation, DigitalModel, ModelManager, DigitalRoad, DigitalRoadSegment, DigitalTrafficLight, DigitalTrafficLoop	SumoSimulator
PPR_2	—	ContextBroker, SubscriptionManager, MongoManager, TimescaleManager
PPR_3	P2DAdapter, D2PAdapter, Adapter, TrafficLightReceiver, TrafficLoopProvider, BolognaPhysicalTwin	Agent, TrafficLightReceiver, TrafficLoopProvider, BolognaPhysicalTwin
PPR_4	ServiceManager, FeedbackTrafficLight, TwinManager	Monitor, Visualizer, Planner, ScenarioGenerator, ServiceManager, TwinHMI, FeedbackTrafficLight, TwinManager
PPR_5	DataManager, DataModel	DataManager, DataModelManager, DataModel, DatabaseManager
PPR_6	ShadowManager, DigitalShadow, RoadShadow, TrafficLoopShadow	ShadowManager, DigitalShadow, RoadShadow, TrafficLoopShadow

ulations, and exploring results.

To enable bidirectional data exchange, FIWARE-specific entities are introduced. The Agent class connects to city data sources and actuators, adapting data formats and storing device IDs and keys. It communicates with ContextBroker, which manages real-time context data, stored in a Mongo database by the MongoManager. Since the Context Broker only holds current states, historical data are managed by the SubscriptionManager and stored in a Timescale database via the TimescaleManager. These two storage mechanisms are required by FIWARE's architectural specifications. For broader data handling, the DatabaseManager abstracts database operations and allows future storage extensions. Lastly, the DataModelManager ensures compliance with the Transportation data model offered by FIWARE.

PIM2PSM Transformation Rules. To automate the mapping of platform independent entities and relations to the Bologna MoDT PSM, six PIM2PSM transformation rules have been defined. These rules describe how abstract model elements are tailored to specific technologies. Along with the PIM model and technology specifications, they serve as input for the second execution of the M2DT tool. As done for CIM2PIM rules, Table 4 outlines each rule, including its identifier, name, and generic input PIM entities, and corresponding output PSM classes. Their application to the Bologna case study is shown in Table 5, where each rule is instantiated using specific entities from the Bologna MoDT PIM.

Rule PPR_1, named transformDigitalModel, merges all digital modeling entities into a unified simulator entity, resulting in the SumoSimulator class. Rules PPR_2 and PPR_3 introduce FIWARE-specific components. Specifically, PPR_2, named createFiwareContext, adds core elements for managing context data and supporting both current and historical storage. Figure 7 illustrates the outcome of rule PPR_2, with the introduced classes and relationships highlighted in yellow. Rule PPR_3, named transformAdapter, introduces the Agent class, replacing separate physical-to-digital and digital-to-physical adapters existing in the previous modeling level with the FIWARE adapter. In our case study, we adopted the same type of FIWARE Agent adapter for both traffic loop providers and traffic light receivers. However, it is possible to define distinct agents for each type of data provider or receiver, depending on the specific MoDT requirements.

Rule PPR_4, named transformService, refines the service layer by extending the generic manager with specific DT service entities. As previously said, Bologna MoDT aims to simulate, monitor and visualize current and historical traffic conditions, therefore the Planner, ScenarioGenerator, Monitor, and Visualizerare included. The TwinHMI is added to centralize user interface functions. Finally, rules PPR_5 and PPR_6 handle data-related aspects: the former introduces database management elements, while the latter integrates the unchanged temporal entities from the PIM into the PSM.

5.5. Implementation Specific Level

This section describes the code artifacts generated during the fourth and final step of our model-driven approach: the model-to-text transformation. This step converts the platform specific model into an implementation specific model, which comprises concrete software components that can be further refined for the final deployment and operational use of the Bologna MoDT.

As explained in Section 5.1, this PSM2ISM transformation is carried out automatically using the code generation feature provided by the Visual Paradigm modeling software. More in detail, we used the *Instant Generator* tool[16], which applies its own set of transformation rules based on the selected programming language. We chose Python language due to its mature ecosystem and strong compatibility with the platforms adopted in our Bologna MoDT. For example, the TraCI (Traffic Control Interface) library is used to easily interface the SumoSimulator class with the deployed SUMO engine [57].

The generated ISM includes 25 Python modules, each corresponding to a class defined in the PSM. Every module adopts

[16] https://www.visual-paradigm.com/support/documents/vpuserguide/124/255/7353_instantgener.html

the same class name, defines the class structure, and includes placeholders to represent the relationships shown in the class diagram. For example, the following listing shows the auto-generated module for the TwinManager class. This module defines three attributes: one for managing multiple instances of shadow managers, one for referencing the simulator class, and one for linking to the services manager. These attributes are automatically derived from the associations defined in the model, where relationship types (e.g., one-to-one, one-to-many) inform the attribute structure (e.g., scalar or list). Additionally, the method included in the class is auto-generated as a basic setter to reflect and manage the corresponding relationship. This method serves as a placeholder, to be refined with application-specific logic during later development phases.

twinmanager.py

```
class TwinManager:
    def __init__(self):
        self.shadow_managers = []
        self.sumo_simulator = None
        self.service_manager = None
    def add_shadow_manager(self, shadow_manager):
        self.shadow_managers.append(shadow_manager)
```

The automatically generated code artifacts are the foundation of the MoDT system, defining class structures, relationships, and basic scaffolding methods that reflect the design model's structure. Even though in our case study the behavioral logic required to make them operational (e.g., coordination mechanisms, or decision processes) is not automatically generated, the overall MDA approach is not undermined. On the contrary, the process ensures that the generated code remains modular, consistent, and traceable across all abstraction levels, thereby supporting efficient development and reducing the risk of design-to-implementation mismatches or manual errors.

6. BoMoDT Platform Deployment and Evaluation

This section presents the deployment and the evaluation of the BoMoDT platform, resulting from the application of the proposed model-driven approach to the city of Bologna, Italy. Specifically, Section 6.1 describes the deployment of the BoMoDT platform, highlighting how the model-driven code artifacts and supporting technologies are integrated to operate the Bologna MoDT. Section 6.2 then details the evaluation of the platform against its functional objectives. Source code and data are available at: https://github.com/sommaalessandra/bomodt.

6.1. BoMoDT Deployment

As discussed in Sec. 5, the structure of the platform' software module was automatically generated from high-level digital twin and mobility specifications, and then application-specific control logic code was added to meet the platform's functional objectives. BoMoDT is designed to model and simulate traffic scenarios, monitor and visualize current and historical traffic flow conditions . BoMoDT facilitates a comprehensive understanding and analysis of traffic dynamics, generating valuable information that can be fed back into the real-world environment and assist human operators in making informed decisions

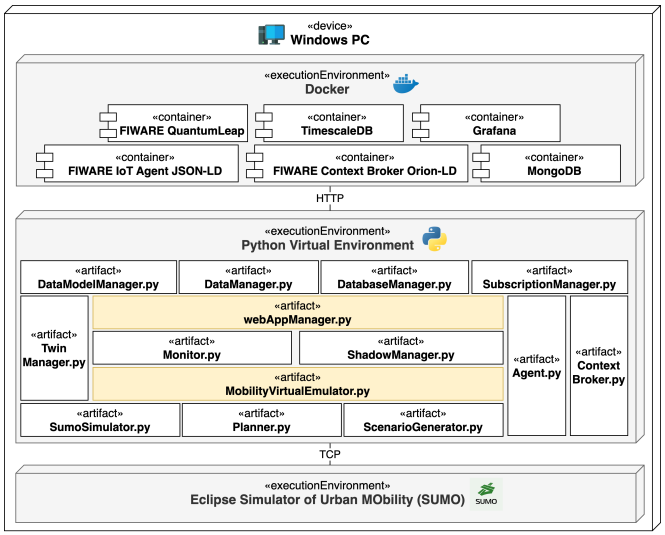


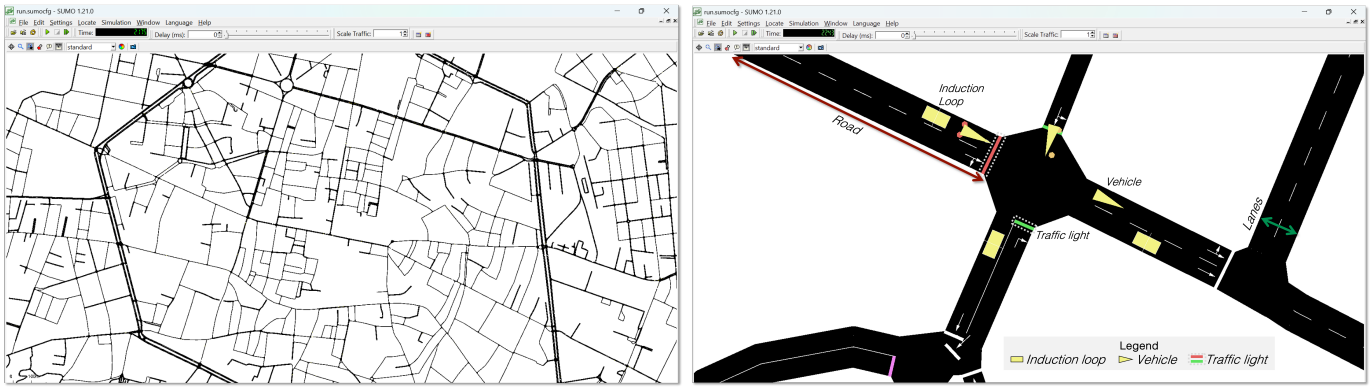
Figure 8: BoMoDT Deployment Diagram.

to improve traffic conditions. As explained in PSM description, predictive features are not included, as they fall outside the scope of this work.

Figure 8 illustrates the deployment diagram of the BoMoDT platform, showing how its software components are distributed across different nodes. The platform runs on a Windows-based device and is structured into three main execution environments:

1. *Docker*. It is used to deploy FIWARE components, which are officially released as Docker images to simplify installation, enhance modularity, and support scalable integration [55]. The deployed containers include:
   - IoT Agent JSON-LD[17], which acts as the interface between the physical environment and the BoMoDT platform. It receives traffic data from devices communicating over standard protocols (e.g., HTTP, MQTT) using JSON-LD format. The agent also manages device registration by validating unique identifiers and secure keys, ensuring only authorized devices can send pre-registered measurements. Device metadata are stored in the MongoDB container. Each HTTP message includes the header Content-Type: application/ld+json to specify the data format and FIWARE-specific headers such as Fiware-Service and Fiware-ServicePath, to support multitenancy for associating each physical device with its key and context data.
   - Orion-LD Context Broker[18], which manages real-time context information. It updates the current state of traffic entities, based on the Transportation data model, in Mongo database, whenever new measurements are received or control commands are issued by the MoDT.
   - QuantumLeap[19], that complements Orion broker by subscribing to context updates and persisting historical data in TimescaleDB.

[17] https://fiware-iotagent-json.readthedocs.io/
[18] https://fiware-orion.readthedocs.io/
[19] https://quantumleap.readthedocs.io/



(a) Bologna represented in SUMO.

(b) City traffic infrastructure.

Figure 9: Digital Modeling of the Italian city.

- Grafana[20], which is used to visualize time-series data from Timescale database through an interactive dashboard, enabling both real-time monitoring and retrospective analysis of traffic conditions.

2. *Eclipse SUMO*. The traffic simulator is deployed in a dedicated environment and communicates with the rest of the platform through a TCP connection using the TraCI protocol. This setup allows for real-time interaction between the simulator and the platform's control logic, for example, to start, pause, or modify the simulation. All required binaries and configuration files are included, enabling SUMO to operate independently while remaining fully integrated into the system workflow. Input files (e.g., the road network, dynamic traffic demand, and traffic light logic) and output files (e.g. traffic statistics or vehicle traces) are structured in XML format and persisted in local storage solutions.
3. *Python*. This environment is responsible for executing the core BoMoDT software modules. It includes both the automatically generated artifacts from the model-driven approach and two additional modules developed to meet specific system requirements. The generated artifacts mirror the structure and associations defined in the platform-specific model and code have been extended with control logic to support interaction with both the containerized components and the SUMO simulation engine. The two additional modules, highlighted in yellow in Fig. 8, are as follows:
   - the mobility virtual environment module, which emulates Bologna's physical traffic infrastructure, compensating for the lack of direct access to real-world sensors and actuators. As a functional DT relies on continuous bidirectional data exchange with its physical counterpart, this module ensures operational realism by sending Bologna real traffic data to the agent and receiving messages and control commands from the MoDT through the broker.
   - the web application manager module, that enables human-machine interaction through a custom dashboard built with the Django[21] framework. This component provides visualization and control features and interfaces with other system modules via REST APIs.

[20] `https://grafana.com/`

[21] `https://www.djangoproject.com/`

6.2. BoMoDT Evaluation

To evaluate BoMoDT functional objectives, we considered two main aspects, namely the ability to run accurate simulations and to respond in a timely manner to system changes. To this aim, we answered the following two research questions (RQs):

RQ1. How accurately can BoMoDT simulate urban mobility conditions using real-world data?
This question focuses on assessing the platform' *simulation fidelity*, using traffic-related metrics to evaluate how closely simulations reflect actual traffic dynamics.

RQ2. How effectively can BoMoDT provide monitoring and visualization of current and past urban mobility conditions?
This question examines the platform's *monitoring responsiveness*, using latency metrics to assess the system's ability to deliver timely traffic updates and visualizations for effective analysis and decision-making.

6.2.1. RQ1: Simulation Fidelity

This RQ aims to evaluate BoMoDT simulations' fidelity, namely, the platform's ability to accurately reproduce real urban mobility conditions using actual traffic data. High simulation fidelity is essential to verify that the behavior of the physical traffic system is realistically reproduced within the MoDT, ensuring that insights and analyses derived from it are accurate and grounded in real-world dynamics. To answer the RQ1, we computed a set of traffic evaluation metrics by comparing the outputs of platform-executed simulations with the corresponding real-world measurements.

As introduced in Sec. 5, BoMoDT operates on traffic flow data collected from 312 induction loop sensors installed across the city of Bologna. These real-world measurements are fed into the platform through the mobility virtual emulator, which streams the data to the FIWARE-based infrastructure and stores them in the platform's persistent database. Once data for a complete hourly time slot (e.g., 08:00–09:00) are available, the twin manager component automatically initiates a simulation. This execution includes the generation of the required input files for

Eclipse SUMO, launching the simulation, storing the results, and rendering them on the platform dashboard.

Each simulation executed by BoMoDT relies on three essential inputs: the road network, the infrastructure configuration, and the traffic demand. The road network was imported from OpenStreetMap[22] (OSM) and manually refined to address typical issues in open-source data, such as missing connections and geometric misalignment. The infrastructure configuration includes the geographical locations of traffic lights and induction loop detectors based on real-world data, to replicate Bologna's actual traffic system. Figure 9 illustrates the digital representation of the Bologna area used for simulation: Fig. 9a renders the city network within the SUMO environment, and Fig. 9b illustrates the modeled infrastructure, including roads, intersections, traffic loops, traffic lights, and vehicles traversing the network.

Last input is the traffic demand which specifies the number of vehicles, their origin-destination pairs, and routes, and it is dynamically generated for each hourly time slot of flow data provided by the emulator. This ensures that every simulation run reflects the actual traffic conditions observed during that period.

The experiments were conducted on a Windows PC with an Intel Core i7 processor, 32 GB RAM, and a 1 TB SSD. For each hourly time slot between January and April 2024, BoMoDT executed a consistent workflow: it received real traffic data from the emulator, ran the simulation, and stored the resulting outcomes. Among the various outputs generated by SUMO, we focused on detector-based flow reports, which capture the number of vehicles passing over each induction loop during a specific time slot. To account for possible stochastic variations in the simulation engine and ensure statistical reliability, each simulation was executed three times. The average of these runs was then used for comparison with actual traffic flow data.

The evaluation was based on three widely recognized metrics: the coefficient of determination (R^2), the Geoffrey E. Havers (GEH) statistic, and the Mean Absolute Percentage Error (MAPE). More in detail, the **coefficient of determination** R^2 quantifies how well the simulation explains the variance in the observed data and is computed as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(O_i - S_i)^2}{\sum_{i=1}^{n}(O_i - \bar{O})^2} \tag{1}$$

where O_i and S_i denote the observed and simulated traffic flows at detector i, respectively, $\bar{O}$ is the mean of the observed values, and n is the total number of detectors. The **GEH** statistic is a domain-specific metric used in traffic engineering to assess how closely simulated volumes align with observed ones. It is calculated for each detector as:

$$GEH_i = \sqrt{\frac{2 \cdot (O_i - S_i)^2}{O_i + S_i}} \tag{2}$$

Values of GEH below 5 are typically considered indicative of strong agreement between observed and simulated data. Lastly,

[22] https://www.openstreetmap.org/

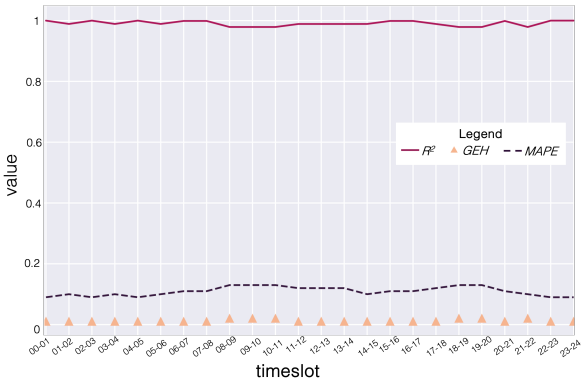


Figure 10: Fidelity metrics for an example day.

the **MAPE** quantity provides a normalized percentage measure of average error across detectors and is computed as:

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{O_i - S_i}{O_i} \right| \tag{3}$$

Figure 10 illustrates a representative example, showing the evaluation metrics over a 24-hour period on March 15th, 2024, aggregated across all detectors. On this specific day, the R^2 values remain consistently close to 1, indicating a strong correlation between simulated and actual traffic flows. GEH values stay well below the commonly accepted threshold of 5 throughout the day, and MAPE values remain low and stable, highlighting a minimal deviation between the simulation and real-world data.

Answer to RQ1: The R^2, GEH, and MAPE results collected over the January–April 2024 period demonstrate a consistent and strong alignment between the simulated and observed traffic flows across all time slots and locations. These results validate BoMoDT *simulation fidelity* and confirm its accuracy in replicating real-world urban mobility dynamics.

6.2.2. RQ2: Monitoring Responsiveness

This RQ focuses on assessing the effectiveness of BoMoDT in monitoring and visualizing both current and historical traffic conditions. In particular, it evaluates the platform's monitoring responsiveness, i.e., its ability to deliver timely traffic data visualizations that support effective analysis and decision-making. Responsiveness is measured in terms of latency, defined as the time elapsed between the reception of traffic flow data by the MoDT and its visualization on the platform dashboard, with or without the inclusion of simulation-generated insights.

Two types of latency were measured, conducting experiments using the same hardware setup previously described. The first, referred to as the **direct** configuration, measures the time elapsed from when BoMoDT receives real-world traffic data from the emulator to when it displays the corresponding traffic indicators on the dashboard, without performing any simulation. The second, referred to as the **enhanced** configuration, measures the time from when BoMoDT acquires the traffic data to the point at which it completes the preparation of simulation

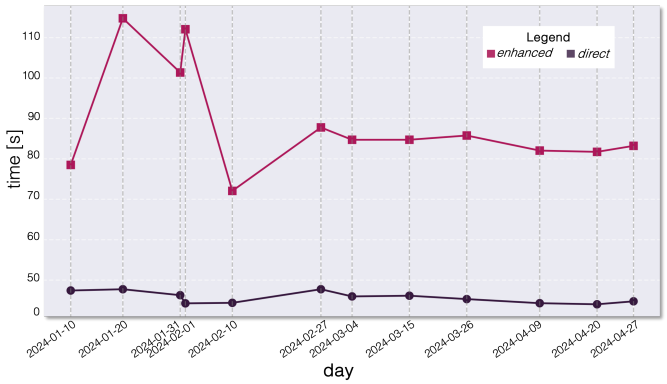


Figure 11: Latency: direct vs. enhanced monitoring.

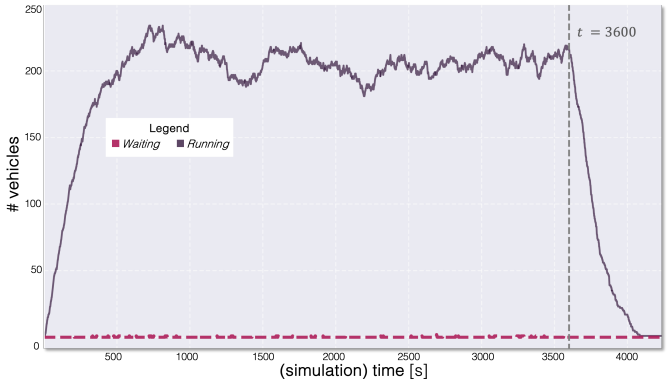


Figure 12: Example simulation run showing vehicles in the network.

inputs, executes SUMO simulation, and visualizes both the current traffic conditions and the simulation outcomes.

Figure 11 compares the latency measurements for the two configurations, based on data collected over twelve representative days between January and April 2024. In the direct configuration, latency remains consistent, with an average of approximately 45 seconds. In contrast, the enhanced configuration shows a latency increase to about 90–100 seconds, mainly due to the extra time needed for simulation execution and result visualization. This increase is directly attributable to the duration of the simulation process.

Figure 12 shows an example of a simulation output for a one-hour time slot, as visualized within the BoMoDT dashboard. The plot depicts the number of vehicles present in the network throughout the simulation: the purple line represents vehicles actively moving, while the dashed fuchsia line indicates vehicles temporarily stopped, typically due to red lights or congestion at intersections. Although the traffic input covers a 3600-second window (equivalent to one real-world hour), the simulation does not stop and continues running until all vehicles introduced have completed their journeys and exited the network. In this specific case, the simulation concludes around 4300 seconds of simulated time, which corresponds to approximately 110 seconds of actual execution time. This extended simulation period is essential to avoid prematurely removing vehicles from the network, which would negatively affect the accuracy of the results. Ensuring all vehicles complete their trips maintains the fidelity of the simulation and preserves consistency with real-world traffic dynamics.

Answer to RQ2: The latency metrics collected from January to April highlight BoMoDT's ability to respond efficiently, with traffic updates and visualizations delivered within acceptable time frames. In the direct setup, the platform provides timely analysis of current traffic conditions, whereas the enhanced configuration introduces some additional delay due to simulations, raising the need to trade off between higher fidelity and responsiveness. Nevertheless, these findings confirm BoMoDT's *responsiveness* in both monitoring and visualization, making it a valuable tool for retrospective mobility analysis.

7. Discussion

This section describes the effectiveness of the proposed model-driven approach, applied to the Bologna case study, in addressing the key challenges associated with MoDTs, discussed in Sec. 3.1. Table 6 summarizes how each challenge is addressed by the model-driven approach or by the enabling technologies. In the table, qualitative coverage is marked as full (✔) or partial (≈). A challenge is: *fully covered* when the approach or technology supports all relevant aspects, even though not all are demonstrated in case study; *partially covered* when only some aspects are addressed, or when additional (manual or automated) activities are needed to achieve full support.

Addressed by Technologies. Interoperability C_2 and scalability C_3 are covered by the chosen technological stack. More specifically, FIWARE supports NGSI-LD standard and open smart data models, which ensure both syntactic and semantic consistency in context data exchange through the broker. Therefore, FIWARE enables full coverage of the *interoperability* challenge. Indeed, even though Bologna case study does not address cross-domain interoperability (e.g., integration with energy systems), FIWARE provides the necessary mechanisms, such as ontology extensions and customizable data models, to support such broader integration.

Scalability is partially addressed by FIWARE's modular and containerized microservice architecture, which allows each component to be independently deployed, replicated, and scaled. Specifically, FIWARE supports distributed deployments, load balancing, and data federation across services, allowing the platform to scale horizontally (by adding more nodes) or vertically (by increasing processing resources) as data volume and velocity increase. However, the overall scalability of a MoDT system also depends on the simulation engine and its ability to process high-throughput data. Simulation components often represent the main bottleneck in DT architectures [32]. In our BoMoDT prototype, SUMO sequentially executes simulations, which significantly limits performance under high data loads and hinders large-scale scalability. Even when the underlying platforms support scalable data infrastructure, C_3 is still constrained by MoDT simulation components.

Table 6: Coverage of MoDT challenges with the proposed approach.

ID	Challenge	Cov.	Addressed by:	Description
C_1	Composability	≈	MDA	Modular framework enable structured integration of other DTs components.
C_2	Interoperability	✔	FIWARE	NGSI-LD and Smart Data Models ensure semantic data exchange.
C_3	Scalability	≈	FIWARE	Microservices and federation support scaling; simulation limits remain.
C_4	Adaptability	✔	MDA	Abstraction layers support easy reconfiguration across contexts.
C_5	Maintainability	≈	MDA	Traceability aids updates; expert knowledge still needed.
C_6	Automatability	✔	MDA	End-to-end model transformation minimizes manual coding.

Addressed by model-driven approach. The proposed approach addresses the remaining challenges, i.e. composability C_1, adaptability C_4, maintainability C_5, and automatability C_6. *Composability* is partially supported through the modular structure of the modeling framework, which separates domain-specific logic from application-independent DT architectural elements. This facilitates the integration of additional lower-level DTs into a unified MoDT model. In our BoMoDT, incorporating a ride-sharing DT would mainly involve updating the domain analysis and modifying related specifications. However, full composability often requires orchestrating multiple simulation engines (e.g., co-simulation), which introduce challenges that are beyond a model-driven approach' scope. These issues are typically case-specific and require additional technological or runtime solutions.

Adaptability is fully supported. The use of modeling layers ensures that changes in the urban context, such as infrastructure updates, or data source modifications, can be addressed by updating high-level models, and automatically propagating changes through the transformation pipeline, minimizing manual rework. Although the operational stability resulting from such changes is not quantitatively measured, the approach inherently fosters stability by confining their impact to specific abstraction layers. *Automatability* is also fully supported. Even though in Bologna case study certain steps remain manual due to their context-specific nature, the proposed approach provides an end-to-end transformation pipeline, automating the transition from conceptual models to executable code artifacts, reducing manual effort, improving consistency, and ensuring alignment across levels.

Lastly, *maintainability* is partially supported. The model-to-code traceability helps keep implementation aligned with design, making it easier to manage updates or extend the system. However, maintaining and evolving models still requires proficiency in modeling notations and possibly transformation logic, which may challenge teams lacking specialized expertise. Additionally, while traceability aids maintenance, the process of identifying which parts of the model must be updated in response to real-world changes can still be complex, especially in large-scale or evolving systems, where ease of maintenance becomes more subjective and context-dependent.

7.1. Benefits and Limitations

As demonstrated by the addressed challenges, the proposed approach provides several benefits in MoDTs development. It enables a structured progression from high-level requirements to code generations through a layered and automated transformation pipeline. This supports generalizability and reusability: if a new platform or technology is adopted (e.g., a different traffic simulator), only lower modeling levels need to be adapted, while higher-level models remain unaffected. Furthermore, the approach is not tied to any specific city or urban context: its design principles can be reused across different domains, such as energy, or healthcare, provided that appropriate domain-specific models are defined. The use of modeling languages and open-source technologies enhances the approach's transferability and accessibility. By clearly separating domain-specific models from DT abstractions, the approach supports collaboration between domain experts and DT engineers, which is essential since DT developers often lack expertise in a specific domain, while domain specialists are typically not trained in technical aspects.

Despite these strengths, the approach presents several limitations. While modular and systematic, the MDA methodology is inherently complex and requires familiarity with modeling notations, meta-modeling principles, and potentially model transformation languages. This complexity may limit its usability for teams lacking MDE experience. To improve accessibility and usability, integrating low-code or agile model-driven extensions could be explored in future work. Moreover, although the generated artifacts preserve traceability and modularity, system evolution and maintenance are not entirely straightforward. Updating models, especially when structural or behavioral requirements change, can be challenging and still requires specialized knowledge. For instance, while structural model updates propagate automatically through the transformation pipeline, any updates that affect transformation logic or domain-specific semantics demand manual intervention and careful validation to maintain consistency across abstraction layers. A key limitation lies in the initial domain modeling step, which can be both time-consuming and error-prone, particularly in multidisciplinary teams. Future improvements could involve developing domain-specific modeling templates or interactive support tools to streamline this phase.

Another practical constraint involves the simulation platform. The chosen simulator significantly affects both the technical integration effort (from PIM to PSM) and the overall system performance. In particular, simulation remains a bottleneck for real-time or large-scale deployments, as performance limitations in simulation tools like SUMO can restrict responsiveness and throughput. Additionally, while the M2DT tool automates the generation of structural code artifacts, behavioral logic must still be manually implemented. As a result, the final code base consists of both auto-generated scaffolding and

manually written control logic, which may introduce inconsistencies and reduce maintainability if not properly managed. Future enhancements could address this by integrating behavioral modeling through dynamic diagrams (e.g., sequence or state diagrams) for comprehensive code generation.

8. Conclusion and Future Work

This paper introduced a model-driven approach for the systematic design and development of MoDTs, addressing key challenges that currently limit their adoption. Based on the Model-Driven Architecture framework, the approach establishes a structured process that spans the MoDT development process through a series of transformations across four abstraction levels. The feasibility and effectiveness of the approach were demonstrated through a real-world case study for Bologna city. This was supported by the developed M2DT tool, which automates the model-driven workflow, and results in the deployment of BoMoDT, a fully operational platform built on open-source data and technologies. The platform's simulation fidelity and monitoring responsiveness were evaluated to assess its capability in accurately simulating, monitoring, and visualizing both current and historical mobility conditions.

The proposed approach has proven effective in addressing key challenges in MoDT engineering, such as adaptability and automatability. The integration of external data management technologies further enhances interoperability and supports scalability. Nevertheless, some limitations remain, especially concerning long-term maintainability and the scalability of simulations under high-load conditions. Future work will aim to advance the approach in three main directions: *(i)* automating the generation of behavioral logic by incorporating modeling of runtime interactions among structural elements, *(ii)* improving scalability to support large-scale and distributed MoDT deployments, and *(iii)* advancing composability to enable the development of integrated Urban Digital Twins. We also plan to quantitatively assess the challenges coverage of the proposed approach, by applying it to different types of MoDTs and involving independent practitioners in diverse use cases. These efforts aim to strengthen the robustness and applicability of the proposal, bridging the gap between conceptual MoDT frameworks and practical, real-world deployments.

9. Data and Code Availability

The **literature analysis**, including the papers obtained from each digital library, the selection process, papers retrieved through snowballing, and the analysis of the selected studies, is available at: https://docs.google.com/spreadsheets/d/1hNPAHboFAdLjr3NsHmt4lbSlIULCiWyF/edit?usp=sharing&ouid=110130922453565990010&rtpof=true&sd=true .

The **M2DT** tool can be accessed via its GitHub repository at: https://github.com/alessandrasomma28/m2dt.

The **BoMoDT** platform is available at: https://github.com/sommaalessandra/bomodt.

Each GitHub repository includes a detailed README file that outlines the design, implementation, deployment, and usage of the respective projects, along with the required input data and source code.

10. Funding

This work has been partially supported by the Spoke 9 Digital Society & Smart Cities of Centro Nazionale di Ricerca in High Performance-Computing, Big Data and Quantum Computing (ICSC), funded by the European Union - NextGenerationEU (PNRR-HPC, CUP: E63C22000980007), by the Swedish Knowledge Foundation through the MoDEV project (20200234), by Vinnova through the iSecure (202301899) and AIDA (202402068) projects, and by the KDT Joint Undertaking through the MATISSE project (101140216).

References

[1] M. S. Irfan, S. Dasgupta, M. Rahman, Toward transportation digital twin systems for traffic safety and mobility: A review, IEEE Internet of Things Journal 11 (14) (2024) 24581–24603. doi:10.1109/JIOT.2024.3395186.

[2] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, P. Tiwari, Mobility digital twin: Concept, architecture, case study, and future challenges, IEEE Internet of Things Journal 9 (18) (2022) 17452–17467. doi:10.1109/JIOT.2022.3156028.

[3] T. Alam, R. Gupta, N. N. Ahamed, A. Ulah, A. Almaghthwi, Smart mobility adoption in sustainable smart cities to establish a growing ecosystem: Challenges and opportunities, MRS Energy & Sustainability (07 2024). doi:10.1557/s43581-024-00092-4.

[4] H. Xu, A. Berres, S. B. Yoginath, H. Sorensen, P. J. Nugent, J. Severino, S. A. Tennille, A. Moore, W. Jones, J. Sanyal, Smart mobility in the cloud: Enabling real-time situational awareness and cyber-physical control through a digital twin for traffic, IEEE Transactions on Intelligent Transportation Systems 24 (3) (2023) 3145–3156. doi:10.1109/TITS.2022.3226746.

[5] C. Weil, S. E. Bibri, R. Longchamp, F. Golay, A. Alahi, Urban digital twin challenges: A systematic review and perspectives for sustainable smart cities, Sustainable Cities and Society 99 (2023) 104862. doi:10.1016/j.scs.2023.104862.

[6] P. Bellini, S. Bilotta, E. Collini, M. Fanfani, P. Nesi, Mobility and transport data for city digital twin modeling and exploitation, in: 2023 IEEE International Smart Cities Conference (ISC2), 2023, pp. 1–7. doi:10.1109/ISC257844.2023.10293300.

[7] E. Faliagka, E. Christopoulou, D. Ringas, T. Politi, N. Kostis, D. Leonardos, C. Tranoris, C. P. Antonopoulos, S. Denazis, N. Voros, Trends in digital twin framework architectures for smart cities: A case study in smart mobility, Sensors 24 (5) (2024). doi:10.3390/s24051665.

[8] S. M. Mostaq Hossain, S. Kumar Saha, S. Banik, T. Banik, A new era of mobility: Exploring digital twin applications in autonomous vehicular systems, in: 2023 IEEE World AI IoT Congress (AIIoT), 2023, pp. 0493–0499. doi:10.1109/AIIoT58121.2023.10174376.

[9] R. Klar, N. Arvidsson, V. Angelakis, Digital twins' maturity: The need for interoperability, IEEE Systems Journal 18 (1) (2024) 713–724. doi:10.1109/JSYST.2023.3340422.

[10] S. Calzati, No longer hype, not yet mainstream? recalibrating city digital twins' expectations and reality: a case study perspective, Frontiers in Big Data 6 (2023). doi:10.3389/fdata.2023.1236397.

[11] J. Michael, M. Schwammberger, A. Wortmann, Explaining cyberphysical system behavior with digital twins, IEEE Software 41 (1) (2024) 55–63. doi:10.1109/MS.2023.3319580.

[12] F. Bonetti, A. Bucchiarone, J. Michael, A. Cicchetti, A. Marconi, B. Rumpe, Digital twins of socio-technical ecosystems to drive societal change, in: Proceedings of the ACM/IEEE 27th International Conference

on Model Driven Engineering Languages and Systems, MODELS Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 275–286. doi:10.1145/3652620.3686248.

[13] R. Jayaraman, D. Lehner, S. Klikovits, M. Wimmer, Towards generating model-driven speech interfaces for digital twins, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2023, pp. 459–464. doi:10.1109/MODELS-C59198.2023.00080.

[14] H. S. Govindasamy, R. Jayaraman, B. Taspinar, D. Lehner, M. Wimmer, Air quality management: An exemplar for model-driven digital twin engineering, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 229–232. doi:10.1109/MODELS-C53483.2021.00040.

[15] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Vol. 3, Morgan & Claypool, 2017. doi:10.2200/S00751ED2V01Y201701SWE004.

[16] J. Whittle, J. Hutchinson, M. Rouncefield, The state of practice in model-driven engineering, IEEE Software 31 (3) (2014) 79–85. doi:10.1109/MS.2013.65.

[17] A. Brown, Model driven architecture: Principles and practice, Software and System Modeling 3 (2004) 314–327. doi:10.1007/s10270-004-0061-2.

[18] I. Poernomo, The meta-object facility typed, in: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 1845–1849. doi:10.1145/1141277.1141710.

[19] D. Batory, Multilevel models in model-driven engineering, product lines, and metaprogramming, IBM Systems Journal 45 (3) (2006) 527–539. doi:10.1147/sj.453.0527.

[20] E. Felix, D. Lopes, O. S. Jr., A framework based on model driven engineering and model weaving to support data-driven interoperability for smart grid applications, in: Proceedings of the 2020 European Symposium on Software Engineering, ESSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 30–36. doi:10.1145/3393822.3432341.

[21] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, A. Lindow, Model transformations? transformation models!, in: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS'06, Springer-Verlag, Berlin, Heidelberg, 2006, p. 440–453. doi:10.1007/11880240_31.

[22] O. M. Group, OMG model driven architecture (MDA) MDA guide rev. 2.0, accessed: 2024-09-05 (2014).
URL https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

[23] R. Soley, et al., Model driven architecture, OMG white paper 308 (308) (2000) 5.

[24] F. R. di Torrepadula, A. Somma, A. De Benedictis, N. Mazzocca, Smart ecosystems and digital twins: An architectural perspective and a FIWARE-based solution, IEEE Software 42 (2) (2025) 38–46. doi:10.1109/MS.2024.3518752.

[25] P. M. Kuruppuarachchi, S. Rea, A. McGibney, Trusted and secure composite digital twin architecture for collaborative ecosystems, IET Collaborative Intelligent Manufacturing 5 (1) (2023) e12070. doi:10.1049/cim2.12070.

[26] P. Kuruppuarachchi, S. Rea, A. McGibney, An architecture for composite digital twin enabling collaborative digital ecosystems, in: 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2022, pp. 980–985. doi:10.1109/CSCWD54268.2022.9776073.

[27] A. Somma, A. D. Benedictis, M. Zappatore, C. Martella, A. Martella, A. Longo, Digital twin space: The integration of digital twins and data spaces, in: 2023 IEEE International Conference on Big Data (BigData), 2023, pp. 4017–4025. doi:10.1109/BigData59044.2023.10386737.

[28] F. Abbasi, P. Brimont, C. Pruski, J.-S. Sottet, Understanding semantic drift in model driven digital twins, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 419–430. doi:10.1145/3652620.3688256.

[29] F. Dembski, U. Wössner, M. Letzgus, M. Ruddat, C. Yamu, Urban digital twins for smart cities and citizens: The case study of herrenberg, germany, Sustainability 12 (6) (2020). doi:10.3390/su12062307.

[30] J. Conde, J. Muñoz, A. Alonso, S. Lòpez-Pernas, J. Salvachua, Modeling digital twin data and architecture: A building guide with FIWARE as enabling technology, IEEE Internet Computing PP (2021) 1–1. doi:10.1109/MIC.2021.3056923.

[31] A. De Benedictis, F. Rocco di Torrepadula, A. Somma, A digital twin architecture for intelligent public transportation systems: A FIWARE-based solution, in: Web and Wireless Geographical Information Systems, Springer Nature Switzerland, Cham, 2024, pp. 165–182.

[32] L. Kanigolla, G. Pal, K. Vaidhyanathan, D. Gangadharan, A. Vattem, Architecting digital twin for smart city systems: A case study, in: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), 2024, pp. 326–334. doi:10.1109/ICSA-C63560.2024.00061.

[33] S. Thelen, F. Eder, M. Melzer, D. Weber Nunes, M. Stadler, C. Rechenauer, M. Obergrießer, R. Jubeh, K. Volbert, J. Dünnweber, A slim digital twin for a smart city and its residents, in: Proceedings of the 12th International Symposium on Information and Communication Technology, SOICT '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 8–15. doi:10.1145/3628797.3628936.

[34] B. Kitchenham, P. Brereton, A systematic review of systematic review process research in software engineering, Information and Software Technology 55 (12) (2013) 2049–2075. doi:10.1016/j.infsof.2013.07.010.

[35] T. Greenhalgh, R. Peacock, Effectiveness and efficiency of search methods in systematic reviews of complex evidence: Audit of primary sources, BMJ (Clinical research ed.) 331 (2005) 1064–5. doi:10.1136/bmj.38636.593461.68.

[36] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA, 2014. doi:10.1145/2601248.2601268.

[37] A. Bucaioni, R. Eramo, L. Berardinelli, H. Bruneliere, B. Combemale, D. E. Khelladi, V. Muttillo, A. Sadovykh, M. Wimmer, Multi-Partner Project: A Model-Driven Engineering Framework for Federated Digital Twins of Industrial Systems (MATISSE), 2025, p. 1.

[38] E. Ferko, A. Bucaioni, P. Pelliccione, M. Behnam, Analysing interoperability in digital twin software architectures for manufacturing (2023) 170–188.

[39] ISO 23247: Automation systems and integration — digital twin framework for manufacturing, Standard, ISO (10 2021).
URL https://www.iso.org/standard/75066.html

[40] E. Ferko, A. Bucaioni, P. Pelliccione, M. Behnam, Standardisation in digital twin architectures in manufacturing (2023) 70–81.

[41] M. Poursoltan, M. K. Traore, N. Pinède, B. Vallespir, A digital twin model-driven architecture for cyber-physical and human systems, in: Enterprise Interoperability IX, Springer International Publishing, Cham, 2023, pp. 135–144.

[42] T. Brockhoff, M. Heithoff, I. Koren, J. Michael, J. Pfeiffer, B. Rumpe, M. S. Uysal, W. M. P. Van Der Aalst, A. Wortmann, Process prediction with digital twins, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 182–187. doi:10.1109/MODELS-C53483.2021.00032.

[43] P. Muñoz, J. Troya, A. Vallecillo, Using UML and OCL models to realize high-level digital twins, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 212–220. doi:10.1109/MODELS-C53483.2021.00037.

[44] M. Dalibor, J. Michael, B. Rumpe, S. Varga, A. Wortmann, Towards a model-driven architecture for interactive digital twin cockpits, in: Conceptual Modeling, Springer International Publishing, Cham, 2020, pp. 377–387.

[45] D. Lehner, A. Garmendia, M. Wimmer, Towards flexible evolution of digital twins with fluent apis, in: 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021, pp. 1–4. doi:10.1109/ETFA45728.2021.9613222.

[46] A. Fend, D. Bork, CPSAML: a language and code generation framework for digital twin based monitoring of mobile cyber-physical systems, in: Proceedings of the 25th International Conference on Model

Driven Engineering Languages and Systems: Companion Proceedings, MODELS '22, Association for Computing Machinery, 2022, p. 649–658. doi:10.1145/3550356.3563134.

[47] J. Michael, A. Wortmann, Towards development platforms for digital twins: A model-driven low-code approach, in: Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems, Springer International Publishing, Cham, 2021, pp. 333–341.

[48] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, A. Wortmann, Model-Driven Development of a Digital Twin for Injection Molding, Springer International Publishing, Cham, 2020, pp. 85–100. doi:10.1007/978-3-030-49435-3_6.

[49] A. Somma, D. Amalfitano, A. D. Benedictis, P. Pelliccione, Twinarch: A digital twin reference architecture (2025). arXiv:2504.07530.

[50] N. Bicocchi, M. Fogli, C. Giannelli, M. Picone, A. Virdis, Requirements and design architecture for digital twin end-to-end trustworthiness, IEEE Internet Computing 28 (4) (2024) 31–39. doi:10.1109/MIC.2024.3376439.

[51] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, M. Wimmer, Digital twin platforms: Requirements, capabilities, and future prospects, IEEE Software 39 (2) (2022) 53–61. doi:10.1109/MS.2021.3133795.

[52] International standard for software, systems and enterprise–architecture description, Standard, IEEE/ISO/IEC (11 2022). doi:10.1109/IEEESTD.2022.9938446.

[53] L. Bieker, D. Krajzewicz, A. Morra, C. Michelacci, F. Cartolano, Traffic simulation for all: A real world traffic scenario from the city of bologna, in: Modeling Mobility with Open Data, Springer International Publishing, Cham, 2015, pp. 47–60.

[54] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, E. Wießner, Microscopic traffic simulation using SUMO, in: The 21st IEEE International Conference on Intelligent Transportation Systems, IEEE, 2018.

[55] F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng, E. Kovacs, A standard-based open source iot platform: FIWARE, IEEE Internet of Things Magazine 2 (3) (2019) 12–18. doi:10.1109/IOTM.0001.1800022.

[56] M. Bauer, F. Cirillo, J. Fürst, G. Solmaz, E. Kovacs, Urban digital twins – a FIWARE-based model, at - Automatisierungstechnik 69 (12) (2021) 1106–1115. doi:doi:10.1515/auto-2021-0083.

[57] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, J.-P. Hubaux, Traci: An interface for coupling road traffic and network simulators, in: Proceedings of the 11th Communications and Networking Simulation Symposium, Association for Computing Machinery, 2008, p. 155–163. doi:10.1145/1400713.1400740.