JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Semantic Drift Management in Digital Twins

Faima Abbasi[*†], **Jean-Sébastien Sottet**[†], **and Cedric Pruski**[†]
[†]Luxembourg Institute of Science and Technology, Luxembourg
[*]FSTM, University of Luxembourg, Luxembourg

ABSTRACT Digital Twin (DT) technology creates layered digital representations of physical system, asset or process, denoted as actual twin (AT). DT integrates heterogeneous data, models and semantic technologies, enabling monitoring, simulation, prediction, and optimization to enhance decision-making and efficiency, ensuring an accurate and dynamic representation of their AT. The rise of the internet of things (IoT) and cyber-physical systems (CPSs) has generated massive volumes of data, which require contextual information about their source and meaning to extract actionable insights. DTs address this need by unifying system data and behavior into coherent, multi-layered heterogeneous models. However, as real-world conditions and AT evolve, semantic drift emerges, leading to a progressive divergence between the DT and its AT, reducing reliability and effectiveness. Semantic drift often arises from inconsistent evolution and misalignment among heterogeneous models within the multi-layered paradigm, causing semantic mismatches, inconsistencies, and synchronization problems. Correcting semantic drift involves structural and behavioral adaptations across heterogeneous models and current methods typically involve manual updates, which are time-consuming, error-prone, and risk compromising data integrity. In this paper, we overcome these challenges and propose a systematic approach to manage semantic drift in multi-layered, model-driven DTs, supporting structural adaptation across heterogeneous models to ensure semantic consistency. We adopt a three-step approach: (i) identification, (ii) evaluation or measurement, and (iii) propagation, to manage semantic drift systematically. First, we identify and assess variants of semantic drift arising from data. We then propagate the changes across heterogeneous models to effectively correct semantic drift. Finally, we employ a case-based generalization strategy to illustrate our approach, showing how insights derived from specific use case support semantic drift management in broader contexts.

KEYWORDS Digital Twin, Heterogeneous Models, Semantic Drift

1. Introduction

Digital transformation is reshaping industries at an accelerating pace, driven by emerging technologies. Among these, DTs have gained attention as a key enabler (Jeong et al. 2022). DT generates virtual representations of physical systems, enabling real-world processes to be analyzed, optimized, and forecasted through simulation (Fuller et al. 2020). Developing effective DTs requires integrating advanced technologies, including machine learning, modeling and simulation, data analytics, and visualization tools (Fuller et al. 2020). Numerous definitions of DTs exist in literature (VanDerHorn & Mahadevan 2021), with interpretations often varying according to the specific application domain (Thelen et al. 2022; Barricelli et al. 2019). DTs are usually considered as complex software systems and consist of multiple and diverse software artefacts, varying upon specific usecase. Hence intense developmental efforts are needed for designing, developing and operating DTs along with communication and cooperation between software and domain experts (Lehner et al. 2025, 2023; Dalibor et al. 2022). In literature, model-driven engineering (MDE) is commonly adopted to handle system complexity and enhance communication through the use of abstract models. These models are designed to be human-understandable while remaining machine-readable and executable representations of real-world entities and systems

JOT reference format:
Faima Abbasi, Jean-Sébastien Sottet, and Cedric Pruski. *Semantic Drift Management in Digital Twins.* Journal of Object Technology. Vol. 25, No. 3, 2026.
http://dx.doi.org/10.5381/jot.2026.25.3.a7

(Brambilla et al. 2017). DTs are strongly related to models and MDE. Each DT incorporates one or more system models, whose characteristics depend on the system type and intended purpose (Bordeleau et al. 2020). These models may serve descriptive, prescriptive, or predictive roles and can include design, behavioral, scientific, or data-driven representations. As DTs often span multiple engineering domains, their development involves coordinating and managing a diverse set of heterogeneous models. In this setting, heterogeneous models are not only a distinguishing characteristic of a specific type of DT, but rather essential artifacts that drive the engineering, operation, and evolution of DTs (Dalibor et al. 2022; Lehner et al. 2025). Designed for specific purposes, such as building energy optimization, a DT integrates heterogeneous models to interpret data, compute metrics, and provide services like monitoring, fault detection, analytics, and what-if simulations (Eramo et al. 2021; Robles et al. 2023). Acting as a semantic layer, it links real-time CPS data to system metadata for informed decision-making (Hribernik et al. 2021), with schemas, current-state snapshots, and historical data evolving alongside the AT (Lehner et al. 2021; Abbasi, Brimont, et al. 2024).

DTs, like other software artifacts, are dynamic and evolve throughout their lifecycle to meet shifting needs and requirements. Their evolution typically involves three primary types of modifications (Mertens et al. 2024): (i) system-driven updates: changes in the AT or its environment (e.g., new sensors, structural changes, external data integration) (Abbasi, Brimont, et al. 2024), (ii) model refinement: enhancements to improve the DT's accuracy for existing objectives, often leveraging improved data quality or more advanced modeling techniques (Sottet et al. 2025), and (iii) purpose evolution: redefining or expanding the DT's goals (e.g., from thermal comfort to also include energy optimization and air quality), leading to new functionalities and services(Mertens et al. 2024). While much of the literature centers around system-driven updates (structural or data-level changes) (Chevallier et al. 2020; Lehner et al. 2021), whereas a more insidious and underexplored phenomenon exists, termed as semantic drift. *Semantic* refers to the meaning and interpretation of the information represented in heterogeneous models, including the concepts, relationships, and constraints that define how models behave and are understood, rather than just their syntax or execution semantics (Harel & Rumpe 2004). In model-driven DTs, semantic drift arises when the digital representation gradually diverges from its physical counterpart in meaning, behavior, or interpretation. This misalignment may stem from sensor degradation (Lu et al. 2020), modeling limitations (Tzachor et al. 2022), or evolving stakeholder expectations and domain semantics (Hribernik et al. 2021). Unchecked, semantic drift undermines the DT's reliability, especially in real-time diagnostics or control tasks, necessitating continual recalibration. Addressing semantic drift through coordinated, model-aware adaptation helps ensure that DT maintain consistent and correct meaning over time, preserving their trustworthiness and alignment with intended goals as the underlying AT and data evolve. Crucially, semantic drift is not independent of evolution, it is often the result of failed or incomplete model evolution (Abbasi et al. 2025; Abbasi, Brimont, et al. 2024). For example, when a DT's purpose changes or expands, but its architecture or services are not updated accordingly, conceptual misalignment emerges. Likewise, structural schema updates that neglect to transform dependent data layers propagate inconsistencies, further fueling drift. Therefore, semantic drift can be viewed as the consequence of misaligned, in-synchronized, or insufficient evolution management across heterogeneous models. This challenge is exacerbated by the interdependent nature of multi-abstraction layer DT (see Figure 1). Changes to the schema often cascade, requiring coordinated co-evolution of snapshots and historical data to maintain semantic integrity. Since time-series data conforms to the snapshot structure, any schema modification may necessitate structural data transformations. These dynamics echo traditional challenges in model and database co-evolution (Chevallier et al. 2020; Lehner et al. 2021), but are magnified in DTs due to real-time constraints and bidirectional synchronization. Despite the growing availability of DT platforms (e.g., *Microsoft Azure* [1], *AWS IoT Twin-Maker* [2], *Eclipse Ditto* [3], *Fiware* [4], *GreyCat* [5]), most tools provide limited support for detecting or mitigating semantic drift (Abbasi et al. 2025), particularly for coordinated schema evolution and data co-adaptation. Consequently, updates are often implemented manually, increasing risks of data loss, inconsistency, and functional degradation (Lehner et al. 2021). Ensuring semantic fidelity in evolving DTs therefore requires modular, adaptable architectures that enable synchronized evolution across layers.

We made three contributions (Abbasi, Brimont, et al. 2024; Abbasi et al. 2025, 2026) to address our long term goal of semantic drift management in DTs. First, we identified four levels of abstraction in multi-layered DT (see Figure 1), which can be mapped to system design and creation phase of DT life cycle, i.e., (i) **data:** represents information related to data sources, (ii) **models:** represents an abstract system, (iii) **metamodels/schema:** uses a specific domain to define the structure and semantics of models, and (iv) **ontologies:** at a higher level, they represent the shared understanding of concepts and relationships within a domain. These elements constitute heterogeneous models, as they differ in abstraction level, representation, and purpose, ranging from structural definitions to semantic descriptions. Then we employ usecase driven approach considering urban mobility usecase to characterize and map different variants of semantic drift. (Abbasi, Brimont, et al. 2024). Second, we assess the effectiveness of different DT frameworks in *identifying, characterizing and provide any kind of support support for managing semantic drift*. We identify requirements for managing semantic drift through a mobility usecase and illustrative scenarios. Based on these requirements, we establish evaluation criteria to assess language and data specific DT frameworks and examine their effectiveness in handling semantic drift (Abbasi et al. 2025). Third, we provided an approach for heterogeneous model alignment that enables heterogeneous models to interop-

[1] https://azure.microsoft.com/en-us/products/digital-twins
[2] https://aws.amazon.com/iot-twinmaker/
[3] https://eclipse.dev/ditto/
[4] https://www.fiware.org/2022/11/14/extended-digital-twin-for-smart-building/
[5] https://datathings.com/

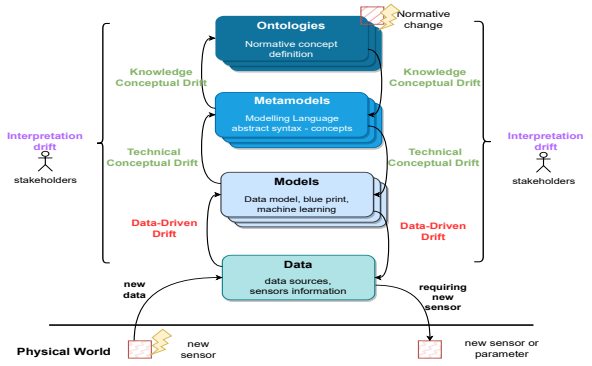

Figure 1 Semantic Drift in DT - *Red* implies change in data; *Green* implies consequent variant.

erate reliably and transparently. It provides a way to design and align abstraction layers in multi-layered, model-driven DTs (see Figure 1) and is based around two core concepts: (i) **flexibility:** we exploit capabilities of *JavaScript modeling framework (JSMF)* to design model and metamodel layer of DT, supporting both rigid and relaxed conformance, while enabling the metamodel to be refined, extended, and continuously updated; and (ii) **alignment:** we proposed a semantic and structure-aware metamodel–ontology matching method (`SSM-OM`) to align metamodel entities with ontology concepts establishing **equivalent to** correspondences, ensuring consistent and domain-relevant DT representations . We used `Brick ontology` [6] for aligning metamodel (Abbasi et al. 2026).

In this work, assuming that heterogeneous model alignment has been done, such as described in (Abbasi et al. 2026), we propose a semi-automated, tool-independent workflow to manage semantic drift in multi-layered, model-driven DTs. Our contributions are stated as:

1. We employ a bottom-up approach with a case-based generalization strategy to systematically analyze and manage different variants of semantic drift in model-driven, multi-layered DTs, emerging from data.

2. We provide methods to identify different variants of semantic drift and propagate changes to correct semantic drift across abstraction layers, exploiting features and constructs of heterogeneous model alignment approach in (Abbasi et al. 2026).

3. We provide different methods to quantify or evaluate variants of semantic drift across each abstraction layer.

Reproducibility of experiments can be found here [7]. Rest of paper covers: (i) background study in section 2, (ii) DT usecase in section 3, (iii) systematic approach for semantic drift management in section 4, and (iv) discussion and conclusion in section 5 and 6.

[6] https://brickschema.org/
[7] https://github.com/faimaAbbasi/dt-semantic-drift-management

2. Background Study

This section reviews the most relevant work on semantic drift management in multi-layered, model-driven DTs, highlighting key contributions and limitations. It then outlines the main challenges in managing semantic drift in DT frameworks and approaches, finally introducing the essential preliminaries and variants underpinning this study.

2.1. Related Work

MDE is a software development paradigm, where domain-specific models serve as primary artifacts across throughout all stages of a system's lifecycle (Brambilla et al. 2017). By capturing domain knowledge and system behavior at a higher level of abstraction, these models enable stakeholders to focus on conceptual understanding rather than low-level implementation details, improving clarity and communication throughout development (Whittle et al. 2013). MDE offers a structured foundation for the evolution and maintenance of DTs. This subsection introduces the two core elements: (i) model management and (ii) consistency management, each of which can help to mitigate semantic drift in multi-layered, model-driven DTs.

2.1.1. Model Management: Effective model management in DT ecosystems is crucial for maintaining consistency and integrity between a DT and its real-world asset. DTs depend on models that reflect their physical counterparts, all structured and interpreted through underlying metamodels. As both the AT and their digital representations evolve, the coordinated evolution of models and metamodels becomes vital. This continuous adaptation encompasses: (i) model evolution, where models are refined, extended, or replaced to maintain accuracy and relevance, (ii) metamodel evolution, where the underlying modeling language or schema is updated to capture emerging system concepts, and (iii) model–metamodel co-evolution, which coordinates changes across both layers to preserve coherence and alignment between abstraction levels. To support this dynamic process, robust model-management practices are required that facilitate smooth migration, effective version control, full traceability, and flexibility, as well as automated runtime updates. Such capabilities should ensure not only synchronized evolution between the AT and its DT, but also coordinated evolution across modeling layers within a multi-layered DT paradigm (see Figure 1). Despite the critical role of model–metamodel co-evolution in sustaining scalable and long-term adaptability, current DT research predominantly emphasizes AT–DT co-evolution, leaving systematic support for co-evolution across multiple abstraction level (e.g., ontology-metamodel-model co-evolution). However below we discuss state-of-art related to evolution and co-evolution of DT and its AT.

DT Evolution: Existing research focuses on how DT models evolve in response to their physical counterpart, especially within machine-learning contexts. The literature highlights the importance of (i) intelligent and autonomous techniques (Dittler et al. 2022), (ii) simulation-driven approaches (Kannapinn et al. 2024), and (iii) federated learning strategies (Sun et al. 2020). Together, these strands emphasize that continual model

evolution is essential for maintaining adaptability, real-time fidelity, and autonomous decision-making in DT systems. Beyond these categories, additional work grounded in MDE further underscores the importance of model evolution within DT systems. Study in (Alskaif et al. 2025) positions evolution as the central challenge in DT engineering, stressing that DTs must adapt continuously alongside their AT. It classifies evolution triggers, i.e., changes in purpose, context, AT behavior, or the DT itself, and highlights multi-dimensional concerns, including stakeholders, services, data, sensing, communication, and modeling. It advocates DevOps practices (modular architectures, CI/CD, telemetry) for runtime adaptation and emphasizes MDE aspects such as model evolution, AT-DT co-evolution, multi-view consistency, and monitoring, outlining key research opportunities in evolution-aware DT design. The study in (Hellwig et al. 2025) proposes a faceted classification framework for organizing and comparing the diverse models used in DT engineering, synthesizing dimensions such as purpose, abstraction, executability, subject, and lifecycle phase. By consolidating scattered taxonomies from MDE and DT literature, study offer a vocabulary that makes explicit how models change roles over time based on their abstraction and purpose, an issue central to model evolution and co-evolution. Martens et al. (Mertens et al. 2024) present *DarTwin*, an MDE-based notation that supports structured evolution of DT through architectural transformations. Their approach introduces domain-independent transformation patterns designed for scalable progression. This method highlights systematic, model-driven refinement that strengthens composability, interoperability, and goal-oriented adaptability.

Metamodels define the structure and semantics of models, and their evolution is essential for maintaining adaptability, accuracy, and clarity. Although largely overlooked in model-driven DTs, metamodel evolution is crucial for integrating new concepts and requirements, ensuring that models and their corresponding twins remain robust and up to date. Study in (Lehner et al. 2021) explore DT evolution with a particular focus on adapting metamodel, data snapshots, and historical records. Their work highlights the importance of coordinating schema changes with corresponding updates to stored data to avoid inconsistencies or loss of information. By organizing evolution across multiple levels, such as types, attributes, instances, and their associated records, they provide a structured approach to managing change. The study also introduces a fluent API for modifying metamodel at runtime and an evolution engine that supports migration, versioning, and data handling, thereby improving automation and maintaining data integrity throughout the lifecycle of a DT.

DT Co-evolution: Literature only focus on how DT co-evolves with its AT, leaving systematic flow of co-evolution in multi-layered DTs. Study in (Mertens & Denil 2023) argues that continuous validation should drive co-evolution between a DT and its AT. The authors distinguish between model validity (structural correctness range) and instance validity (correctness of calibrated parameters) and propose a workflow that continuously compares simulated and real-world data. When deviations occur, the DT first attempts recalibration (instance-level co-evolution); if unsuccessful, it triggers model revision (structural co-evolution). DT can actively inject control experiments into the AT to collect missing data, addressing the *not-enough-data* issue and sustaining trustworthy co-evolution. Study in (Tong et al. 2024) presents a co-evolutionary DT framework where multiple lifecycle DTs form an interacting network rather than evolving independently. They integrate DTs using an MBSE-based architecture and model their co-evolution through information-theoretic and statistical dynamics metrics (e.g., entropy, alignment, interaction strength) to quantify collective behavior. Each DT co-evolves with its physical counterpart while also influencing other DTs across lifecycle stages. A solid rocket engine case study demonstrates improved cross-stage coordination, real-time stability monitoring, and lifecycle-wide optimization. With respect to our work, we clearly articulate our position. Unlike the multi-lifecycle DT network proposed in this study, we adopt a multi-abstraction-layer DT paradigm in which co-evolution occurs across internal layers and components within a unified DT structure.

2.1.2. Consistency Management: This concept helps maintain alignment, accuracy, and coherence among DT models, abstraction layers, instances, and the AT they represent as all components change over time. Only a few existing studies offer frameworks or methods for preserving and evaluating consistency, and those that do primarily tackle issues like latency and structural integrity. The studies in (Muctadir et al. 2024, 2025) introduce a graph-based (Abbasi et al. 2019, 2022; Muzammal et al. 2020; Abbasi, Muzammal, et al. 2024) consistency management framework for DT systems, designed to maintain coherence among heterogeneous models that evolve frequently. Using *Neo4j* and *Cypher* queries for data extraction, unified storage, and rule-driven validation, their approach synchronizes interrelated models at abstraction level. Validated through case studies, including an autonomous truck docking DT and Xtext grammar definitions, the framework demonstrates how graph structures and MDE principles can automate consistency checks, address cross-domain heterogeneity, and strengthen model synchronization within DT environments. The study in (Liu et al. 2024) proposes an evolutionary concurrent modeling approach for DT process models that manages semantic, temporal, and feature-level consistency. Using a multi-layer structure in which each layer evolves independently yet remains interconnected, the method supports incremental updates and coordinated change propagation, improving adaptability and fidelity in representing real-world processes. The study in (Zhang et al. 2022) introduces a structured, MDE-oriented framework for evaluating the consistency of DT models across geometric, physical, behavioral, and rule dimensions, as well as after model assembly and fusion. Using measurement, experimentation, SysML-based checks, and coupling validation, combined through an AHP scoring method, the approach provides a quantitative way to assess model fidelity. Case studies on a pose-adjustment spreader and an AIT shop-floor show the framework's effectiveness, yielding near-perfect consistency for stable mechanical systems and variable consistency for human-influenced processes.

State-of-the-art studies show that evolution, co-evolution

Abstraction Layer	Semantic Drift Variant	R-Imperatives	Bottom-Up Flow - Air Quality Usecase
Data Layer	Data-Driven Drift [Structural Drift]	Re-configuration, Re-collect	Putting new or changed sensor in placement, data appears and is re-collected
Model Layer	Data-Driven Drift [Structural Drift]	Re-model, Re-calibrate, Re-concile	DT instances are created or re-instantiated at model layer, calibrated with data and reconciled with metamodel layer
Metamodel Layer	Technical Conceptual Drift	Re-model, Re-concile	DT entities should be created or updated and reconciled with ontology layer
Ontology Layer	Knowledge Conceptual Drift	Re-model	DT entities should be aligned or re-aligned with semantic concepts and evolved DT is deployed

Table 1 R-imperatives and Semantic Drift Mapping

and consistency management have been addressed primarily for single DTs or through coordination among multiple DTs across lifecycle stages. However, literature hardly explores systematic flow of evolution and co-evolution in multi-layered DTs. Our originality lies in ensuring multi-layered consistency by integrating DT–AT coupled evolution and using ontology-metamodel-model evolution as foundation to support evolution.

2.2. Core Challenges

The study in (Michael et al. 2025) examines challenges along with solutions from MDE body of knowledge, including model evolution, co-evolution, and multi-paradigm modeling, which are key to manage semantic drift. However, it pays limited attention to data management and semantic technologies; while data management supports both structural and behavioral modeling, semantic technologies remain limited in their support for behavioral modeling in multi-layered DTs. While the literature focuses on DT-AT consistency, challenges such as inter-layer evolution, co-evolution and vertical consistency across abstraction layers are less explored. There is lack of automation of different model management features, i.e., adaptability, versioning, rollback, migration and traceability. Metrics for drift evaluation and quantification are also neglected. Overall, there is currently no unified method to collectively address semantic drift in multi-layered, model-driven DTs.

Building on these insights, our long-term goal is to develop a comprehensive framework that systematically manages all variants of semantic drift in an integrated and cohesive manner. To achieve this objective, in this paper: (i) we provide standardized metrics to quantify and evaluate semantic drift across different abstraction layers in multi-layered, model driven DTs and (ii) we address structural adaptability in terms of flexibility and alignment, in multi-layered, model-driven DT, while addressing vertical consistency in multi-layered, model-driven DTs.

2.3. Preliminaries

Semantic drift, defined as a *sudden, gradual, incremental, or recurring shift in a DT component's structure, precision, value, or meaning*, poses a significant threat to model coherence (Abbasi, Brimont, et al. 2024). A formal definition is provided in Definition 2.1. Such drift disrupts the alignment of heterogeneous digital models, including ontologies, software artifacts, and equations. Maintaining DT fidelity therefore requires continuous updates and robust semantic integration across all modeling layers.

Definition 2.1 ***(Semantic Drift)*** *Suppose that a AT in the real world has a DT with a set of heterogeneous models, metamodels and ontologies, at a time period [0,t]. AT in the real world is connected to its DT through a bi-directional data flow, represented by the relation* $\overset{\mu}{\Leftrightarrow}$*, at a given time period [0,t]. Semantic drift occurs above certain threshold* Φ*, at the time period [t+1] to time stamp [t+m] (m can be any number), when the relation* $AT_{[0,t]} \overset{\mu}{\Leftrightarrow} DT_{[0,t]}$ *changes to* $AT_{[t+1,t+m]} \overset{\Delta\mu}{\Leftrightarrow} DT_{[t+1,t+m]}$*, as the real world evolves.*

Note that we use $\Delta\mu$ to represent semantic drift arising from real-world evolution, which renders AT inconsistent with DT. Because small variations may not qualify as drift, a threshold represented by Φ is applied to determine when drift is considered to have occurred. Φ values lie between 0 to 1, near to 0 means minimal or no relevant change, while near to 1 means drastic change. The remaining notations and semantic drift variants are formally defined and illustrated using a case-based generalization approach in (Abbasi, Brimont, et al. 2024). We distinguish two broad categories of semantic drift: (i) **vertical drift**, which occurs across different DT abstraction layers, and (ii) **horizontal drift**, which occurs within the same abstraction layer [8]. As shown in Figure 1, vertical drift consists of three subtypes: (i) **data-driven drift**, where inconsistencies originate in the data and propagate to the model layer, also encompasses **structural drift**, which occurs when changes in sensor calibration alter the data structure; (ii) **technical conceptual drift**, reflecting misalignment between the model and metamodel layer; and (iii) **knowledge conceptual drift**, involving misalignment or inconsistencies between the metamodel and ontology layer. A typical DT life cycle consists of: (i) system design and creation phase, (ii) manufacturing or production phase, (iii) operational phase and (iv) end-of-life (Pronost et al. 2024). Semantic drift is an important consideration within DT life cycle, emerging primarily during the operational phase. It must then be actively managed during the maintenance or

[8] vertical drift can induce horizontal drift, requiring changes to be propagated to higher abstraction layers to correct the resulting inconsistencies or drift.

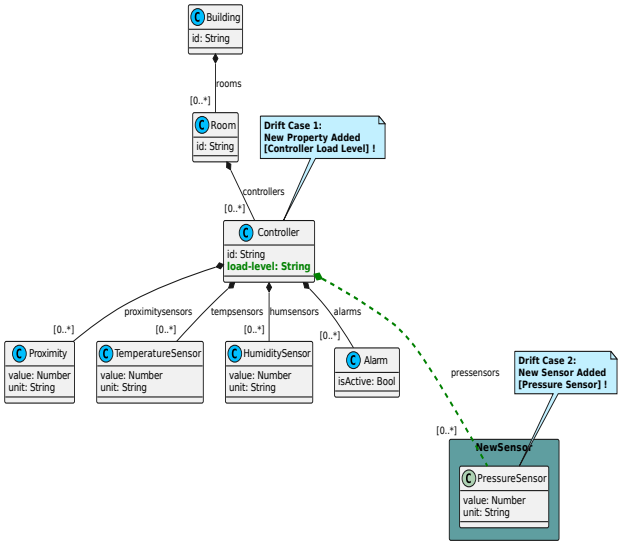

Figure 2 Air Quality Management Exemplar Metamodel

evolution phase, an overlapping sub phase in which the DT is updated, re-calibrated, and refined to maintain fidelity.

Semantic drift is a result of inconsistent evolution. The study in (Michael et al. 2024) presents a 7R taxonomy for DT evolution in smart ecosystems, outlining evolutionary activities that enhance the technical sustainability of DTs. Building on this, it identifies seven corresponding action areas designed to effectively address the evolving needs of DTs and provide R-imperatives. These R-imperatives are discussed in detail in (Michael et al. 2024) and include: (i) re-configuration, (ii) re-collection, (iii) re-modelling, (iv) re-calibration, (v) reconciliation, (vi) re-deployment and (vii) re-use. On the basis of this 7R classification, we orient how semantic drift propagates in bottom up manner, emerging from data and propagating to above abstraction layers, in the context of our air quality usecase depicted in section 3 and how the R-imperatives respond in multi-layered DT. Table 1 illustrates R-imperatives in context to two semantic drift scenarios illustrated in section 3 and Figure 2. We observe that these R-imperatives are confined to the metamodel layer, addressing only technical conceptual drift; knowledge conceptual drift at the ontology layer lies beyond the scope of work in (Michael et al. 2024).

3. Digital Twin Usecase

This section presents the setup of an *air quality management* use case, adopting its basic DT pipeline from (Govindasamy et al. n.d.). The DT supports continuous monitoring, automated alarms, and data-driven optimization. The metamodel (Figure 2) links room controllers with temperature, humidity, and CO_2 sensors and alarms. While we do not have a physical setup with real sensors and actuators, we integrated a 2.5 million-sample *air quality dataset* [9] from 10 sensors (Anik et al. 2025) directly into the model layer. This approach enables the study of semantic drift in a bottom-up manner, where changes are driven by the data itself, with new observations compared against historical records to detect and propagate drift, rather than relying on AT.

Requirements were elicited using a case-based generalization strategy (Wieringa & Daneva 2015), abstracting from one instance to multi-layered DT patterns. The process includes: (i) *use case positioning*: indoor DTs in offices, universities, and hospitals integrate sensor data (temperature, humidity, CO_2) for alarm-based decisions across data, model, metamodel, and ontology layers under real-time and safety constraints; (ii) *modeling abstractions*: domain models define entities (building, room, sensors, alarms) and functional models describe data flow through the four layers; (iii) *requirement elicitation*: structural patterns guide requirements for scalable data streams, adaptive models, extensible metamodels, and evolving ontologies. We focus only on requirements for models, metamodels, and ontologies.

3.1. Semantic Drift Cases

This subsection presents two semantic drift scenarios where data-driven changes trigger structural drift at the model layer, cascading into technical and conceptual drift at the metamodel and ontology layers, requiring propagation across all DT layers to maintain coherence with the real system.

3.1.1. Drift Case 1 - Load Level: A new functionality is added in a controller to measure how intensely a controller is operating based on environmental and occupancy conditions. It helps in performance tracking, balancing, and predictive maintenance. It is calculated using insights from the other sensors, i.e., proximity, humidity and temperature, linked with controller.

```
score = proximity value + (1.5 × (current
    temperature - optimal setpoint)) + (0.5 × (
    current humidity - optimal setpoint))
load-level = "Low" if score < 10 else "Moderate"
    if score <= 20 else "High"
```

Listing 1 Load Level Calculation

A sample load-level attribute calculation is shown in Listing 1, where weights like 1.5 and 0.5 represent the relative impact of environmental factors on the controller's workload.

3.1.2. Drift Case 2 - Pressure Sensor: A new sensor type is added under the *Controller* through *pressensors* association. It holds value and unit properties, enabling the system to monitor air pressure per room. This supports enhanced air quality management by improving sensor calibration, detecting blockages, or managing airflow.

These two drift scenarios are also illustrated in Figure 2, where the controller is extended with a new attribute and an additional sensor. This visual representation emphasizes the evolving structure of the system as it adapts to enhanced monitoring and functional capabilities.

4. Semantic Drift Management

This section presents a concise workflow for managing semantic drift in multi-layered, model-driven DTs. We propose a tool-independent, semi-automated approach that follows a bottom-up

[9] https://doi.org/10.17605/OSF.IO/BAEW7

approach to systematically manage drift emerging from data. Building on the semantic drift variants identified in (Abbasi, Brimont, et al. 2024) and stated in subsection 2.3, we demonstrate how two illustrative cases from subsection 2.3 can be addressed. The focus is on vertical drift, while horizontal drift is beyond the scope of this work. Our workflow includes three steps to manage semantic drift variants: (i) **identification** of changes emerging in data, (ii) **evaluation** or **measurement** of severity of change at each abstraction layer in context to DT subject, and (iii) **propagation** of changes to correct semantic drift across abstraction layers.

4.1. Structural Drift

In this subsection we provide systematic way to identify, evaluate and propagate structural drift.

4.1.1. Drift Identification: We detect structural drift in evolving air-quality data by performing an entity recognition process to identify newly emerging properties and entities. Our approach classifies these elements into predefined entity categories and quantifies the drift between historical and newly observed data. This enables a comprehensive understanding of how the data structure evolves over time and highlights the specific areas where structural changes occur.

We define subject of DT in Listing 2, through a set of general entities $\mathbf{E}=\{\mathbf{e_1}, \mathbf{e_2}, .., \mathbf{e_n}\}$, representing concepts related to our air quality use case, depicted in Figure 2. The subject of a DT can vary depending on the specific use case, reflecting the context or focus of the DT. It essentially defines what the DT is intended to represent or monitor, providing clarity on the system, process, or entity for which the DT is designed [10]. This contextual definition ensures that the DT is purpose-driven and tailored to the requirements of its intended application. This DT subject is defined by DT stakeholders and DT maintainer. Validating the subject of DT is the responsibility of DT maintainer.

Each entity $\mathbf{e_i}$ is associated with set of representative keywords $\mathbf{KW}=\{\mathbf{kw_1}, \mathbf{kw_2}, ..., \mathbf{kw_n}\}$ that describes it, to formulate DT subject. To assign entity $\mathbf{e_i}$ to evolved property $\mathbf{p}$ in data, we compare $\mathbf{p}$ with each $\mathbf{kw_i} \in \mathbf{KW}$ and compute similarity score $S(\mathbf{p}, kw)$ between 0 and 1, where 1 indicates an exact match and 0 indicates no match. Similarity score is computed using ratcliff algorithm[11]. For each entity $\mathbf{e_i}$, we calculate the maximum similarity $S_i = \max_{kw \in KW_i} S(\mathbf{p}, k)$ between the new property $\mathbf{p}$ and all $\mathbf{KW}$ of that entity $\mathbf{e_i}$. The candidate entity $\mathbf{e_i}$ is the one with the highest similarity score S_i, exceeding the threshold represented by τ having value 0.4. τ can be selected by the DT maintainer based on how narrowly or broadly candidate entities are chosen to correspond to the DT subject. Its value lies between 0 to 1. τ near to zero means broad DT subject, while τ near to 1 means narrow DT subject. This can be formally stated in Equation 1.

$$\mathbf{e_i} = \begin{cases} \arg\max_i S_i, & \text{if } \max_i S_i > \tau \\ \text{"Unknown"}, & \text{otherwise} \end{cases} \quad (1)$$

[10] DT subject can also be associated with or inferred from the ontology layer, as it plays a key role in defining concepts that are closely related to ontologies

[11] https://www.npmjs.com/package/string-similarity

We illustrate this phenomenon through the following drift cases, as highlighted in subsection 3.1.

Drift Case 1 - Load Level: Consider the following set of general entities that define a DT subject. For instance, Listing 2 describes the DT subject relevant to our air quality use case as illustrated in section 3, where on one side are entities which are relevant to our metamodel (see Figure 2) and on other side we have keywords acting as synonyms to describe entities, extracted from ontology.

```
const generalEntities = {
  Building:[building, site, location],
  Room:[room, area, zone],
  Controller:[controller, device, system, level],
  TemperatureSensor:[temperature, unit],
  HumiditySensor:[humidity, humidity_unit],
  PressureSensor: [pressure, pressure_unit],
  Alarm:[alarm, status, alert, active]
};
```

Listing 2 DT Subject

Whenever a new property **p="Load Level"** appears in the data according to scenario defined in subsection 3.1, it is compared against each keyword $\mathbf{kw_i} \in \mathbf{KW}$ associated with every entity $\mathbf{e}$ listed in Listing 2 and a similarity score is then computed according to Equation 1. We obtain a similarity score of 0.6, indicating that the emerging property **p="Load Level"** can be associated with the **Controller** entity. We categorize newly evolved properties as property with entity present in the historical data, i.e., $\mathbf{P_{existing\ (e)}}$.

Drift Case 2 - Pressure Sensor: Suppose a new pressure sensor is integrated into the AT and linked with **Controller**, generating corresponding data in the form of new properties (**p="value" and p="unit"**). The first step is for the DT maintainer to validate whether this incoming data aligns with the intended subject and scope of the DT. Based on the defined DT subject given in Listing 2, the emerging data can be associated with the `PressureSensor` entity, with similarity scores of 0.6 and 0.7 validated through Equation 1. We categorize newly evolved properties as property with entity absent in the historical data its concept or entity present in DT subject, i.e., $\mathbf{P_{new\ (e)}}$.

If any property $\mathbf{p}$ is not mapped to set of general entities $\mathbf{E}$ under threshold $\tau = 0.4$, it is considered as unknown. We categorize this unknown property as $\mathbf{P_{unknown\ (e)}}$.

4.1.2. Drift Measurement: We measure degree of structural change using structural drift score in Equation 2, adapted and build upon from (Etien & Anquetil 2024; Stavropoulos et al. 2016), where they provide insights for a weighted composite drift measure, penalizing severe drift (new concepts) more heavily than minor (attribute-level) drift. We categorize newly evolved properties as: (i) $\mathbf{P_{new\ (e)}}$: property with entity absent in the historical data, but present in DT subject, (ii) $\mathbf{P_{existing\ (e)}}$: property with entity present in the historical data and present in DT subject and (iii) $\mathbf{P_{Unknown}}$: properties that cannot be directly inferred from general entities. We use these categories of newly evolved properties to quantify structural drift illustrated in Equation 2.

$$\text{Drift Score} = \begin{cases} \frac{|P_{\text{new}(e)}| + |P_{\text{Unknown}}| + 0.5\,|P_{\text{existing}(e)}|}{|P|}, & \text{if } |P| > 0 \\ \text{no structural drift}, & \text{otherwise} \end{cases} \quad (2)$$

Where $|\mathbf{P}|$ denote the total number of new properties. Among them, $|\mathbf{P}_{\text{new}(e)}|$ are number of properties whose entities do not appear in historical data, these reflect significant structural changes and contribute fully to the drift score (weight = 1). The set $|\mathbf{P}_{\text{existing}(e)}|$ includes number of properties whose entities do appear in historical data, these indicate milder changes and contribute partially (weight = 0.5). Finally, $|\mathbf{P}_{\text{Unknown}}|$ contains number of properties that cannot be inferred from known entities through similarity checks and therefore contribute fully (weight = 1). $\mathbf{P}_{\textbf{Unknown}}$ may cause the DT to drift away from its intended subject or use case. The drift score in Equation 2 is calculated as the weighted ratio of new, unknown, and existing properties observed in the data relative to all properties. This metric quantifies how much the DT subject is evolving or diverging, providing the maintainer with guidance on whether these properties should be incorporated into the DT's abstraction layers.

4.1.3. Drift Propagation:

When a new functionality is added to a controller on AT, it is first observed in data, then it impacts model layer than metamodel layer. It is usually introduced as a new property $\mathbf{P}_{\textbf{existing (e)}}$ within the existing **Controller** entity (see Figure 2) at metamodel layer. Although this may seem practical, it actually changes the controller's schema. Such schema changes can spread inconsistencies to other modeling layers, especially in tightly coupled architectures. The same type of schema change occurs when a new device is added on AT, such as a `PressureSensor`, which first appears in the data, due to emergence of $\mathbf{P}_{\textbf{new (e)}}$. Its information is typically added as a new association to the existing **Controller** entity at metamodel layer. In both cases, whether adding new functionality or introducing a new device, the **Controller's** schema is modified. Once structural drift in the data is identified through the entity recognition process depicted in subsection 4.1.1 and evaluated through drift score in Equation 2, the changes should be propagated to the model layer. Affected DT entities must then be re-instantiated, and any new entities must be instantiated in accordance with the metamodel at model layer, ensuring that their structure and semantics are preserved.

4.2. Technical Conceptual Drift

In this subsection we provide systematic way to propagate and evaluate technical conceptual drift.

4.2.1. Drift Propagation - Model Layer:

After identification of structural drift in the data through the entity recognition process depicted in subsection 4.1.1 and evaluated through drift score in Equation 2, in this subsection provides an in-depth analysis how this drift is propagated at model layer using two scenarios illustrated in section 3.1.

Drift Case 1 - Controller Load Level: In the context of our air quality use case (see Figure 2), when a new property emerges in the data for existing entity, i.e., $\mathbf{P}_{\text{existing}(e)}$, it must first be incorporated into the model layer and subsequently reflected in the metamodel layer to maintain consistency and conformance.

JSMF, inspired by the *Eclipse Modeling Framework (EMF)*, is a dynamic modeling platform that supports partial metamodel conformance, dynamic typing, and modular evolution. Its proto-metamodels, on-the-fly customization and posterior metamodel inference allow models to evolve from exploratory to validated forms (Sottet & Biri 2016). We used JSMF for its flexibility, enabling a bottom-up approach to detect and manage semantic drift. Structural changes in the data trigger updates at the model layer by re-instantiating affected entities, with the metamodel class marked as flexible to accommodate drift seamlessly. Listing 3 illustrates the configuration of flexible model–metamodel conformance, by making **Controller** class flexible and re-instantiating it.

```
function update_model(row, attribute, entity) {
    let flexibleClass = null;
    const classMap = {
        Building: MM.Building,
        Room: MM.Room,
        Controller: MM.Controller,
        TempSensor: MM.TempSensor,
        HumSensor: MM.HumSensor,
        Alarm: MM.Alarm,
    };
    if (classMap[entity]) {
        classMap[entity].setFlexible(true);
        flexibleClass = classMap[entity];
    }
    const controller = MM.Controller.newInstance
    ();
    controller.id = row.Controller_id;
    controller[attribute] = row.load_level; //
    new functionality appeared in data
}
```

Listing 3 Setting Flexibility & Re-instantiting DT Entity

Drift Case 2 - Pressure Sensor: In the context of our air quality use case (see Fig. 2), when a new sensor is added to the AT and starts generating data, it must first be integrated into the model layer and subsequently reflected in the metamodel layer to preserve conformance and consistency. To achieve this, we leverage the features provided by JSMF. Specifically, we begin by instantiating a DT for the `PressureSensor` at the model layer, ensuring it conforms to the `Thing` class. We illustrate this in Listing 4.

```
function update_model(row, attributes) {
    const uniqueEntities = [...new Set(attributes
    .map(item => item.entity))][0];
    const newInstance = new JSMF.Thing();
    newInstance.value=parseFloat(row.pressure);
    newInstance.unit=row.pressure_unit;
    classType=newInstance.conformsTo();
}
```

Listing 4 Creating DT Entity Instance (PressureSensor)

4.2.2. Drift Measurement:

After propagation of changes at model layer, in this subsection we provide details how we evaluate technical conceptual drift between model and metamodel layer.

Drift Case 1 - Controller Load Level: After defining the metamodel class as *flexible* and re-instantiating it as illustrated in Listing 3, we assess technical conceptual drift between the model and metamodel layers using metric in Equation 3.

$$\text{Drift Score (c)} = 1 - \frac{|\mathcal{H}(P) \cap \mathcal{I}(P)|}{|\mathcal{H}(P) \cup \mathcal{I}(P)|} \tag{3}$$

where $\mathcal{H}(P)$ and $\mathcal{I}(P)$ are *flexible* class and instance properties. This metric is used to measure the structural change between flexible class and its instance. It leverages set theory and Jaccard similarity to quantify degree of structural change in terms of missing and new properties between instance and metamodel class, offering a reproducible metric for evaluating model integrity within evolving DT, adapted from (Garcés et al. 2009).

Drift Case 2 - Pressure Sensor: After instantiating a DT entity for the `PressureSensor` at the model layer, we verify whether a corresponding class exists at the metamodel layer. If such a class is present, the DT instance is made to conform to it. Otherwise, the metamodel must be updated to introduce a new class.

4.2.3. Drift Propagation - Metamodel Layer:

After the evaluation of technical conceptual drift we provide details in this subsection how changes are propagated at metamodel layer.

Drift Case 1 - Controller Load Level: Following the evaluation of technical conceptual drift, the metamodel layer is systematically refined through archetype-driven metamodel discovery (Sottet & Biri 2016), anchored in foundational strategies such as metamodel inference and schema extraction. A representative excerpt illustrating this metamodel evolution process, when a new property emerges in the data for existing entity, i.e., $\mathbf{P}_{\text{existing}(e)}$, is presented in Listing 5.

```
function update_metamodel(isValid, instance,
    classType, entity){
    if (!isValid) { // boolean, indicating drift
    in structure of class and instance
      const res = discoverer.archetypalDiscovery(
    entity,instance,classType)
      discoverer.updateClass(classType,res)

    } else {
      console.log("No update required.");
    }
}
```

Listing 5 Metamodel Entity Inference (Drift Case 1)

Drift Case 2 - Pressure Sensor: We present a representative excerpt in Listing 6 illustrating metamodel evolution process based on metamodel inference and schema extraction, whenever a new property emerges in the data for new entity, i.e., $\mathbf{P}_{\text{new}(e)}$ (addition of new sensor on AT).

The `archetypalDiscovery()` method infers a metamodel class from a given instance or archetype (Sottet & Biri 2016). It identifies the structural and referential properties of the instance and builds a new metamodel class that reflects them. Optionally, it can update an existing metamodel class to integrate the inferred structure.

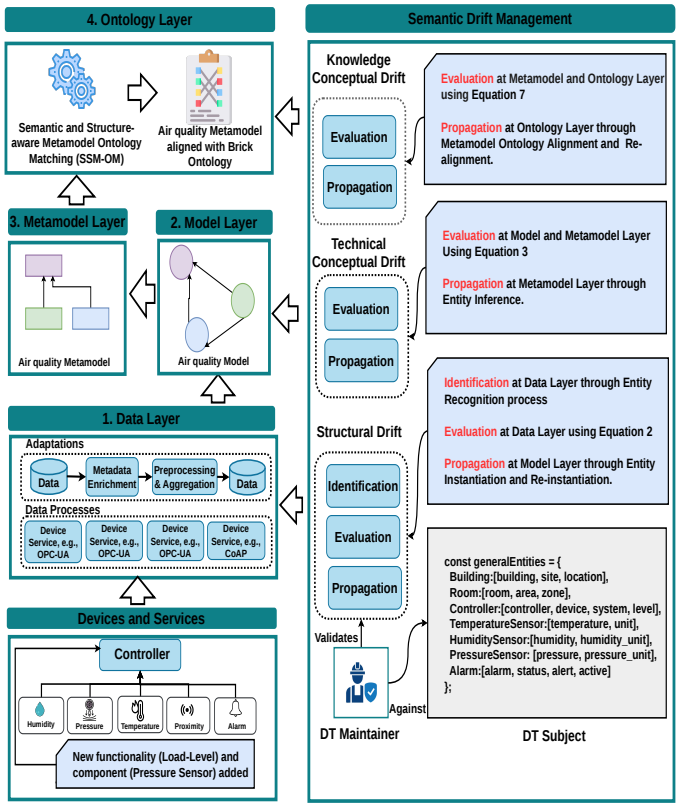

Figure 3 Semantic Drift Management Approach - *Left side*, shows integrated heterogeneous abstraction layers in DT; *Right side*, shows semantic drift management workflow

```
async function update_metamodel(instance,
    classType, entity){
  const cls = getElement(entity) // validates
    class in metamodel
  if (cls == null) {
    const res = discoverer.archetypalDiscovery(
    entity,instance,classType)
    discoverer.updateClass(classType,res)
    add_element(classType, entity)
  } else {
    console.log('cls exists', cls)
  }
}
```

Listing 6 Metamodel Entity Inference (Drift Case 2)

This `archetypalDiscovery()` process can be illustrated through formal notation in Equation 4 and 5.

$$\mathcal{H}(P) = \{(k, \tau(v)) \mid (k, v) \in \mathcal{I}(P)\} \tag{4}$$

Where $(k, v) \in \mathcal{I}(P)$ represents the set of instance properties having k as an attribute key and v as its corresponding value, $\tau(v)$ is the type inference function and $\mathcal{H}(P)$ is the set of inferred attribute–type pairs for updating metamodel class.

$$archetypalDiscovery : \mathcal{I}(P) \rightarrow \mathcal{H}(P) \tag{5}$$

This metamodel evolution step is crucial for enabling real-time generation and adaptation of DT schemas from sensor or asset

instances. It is ideal for dynamic systems with unpredictable data, where manual metamodeling isn't feasible. By supporting on-the-fly schema updates, it helps prevent semantic drift and keeps the DT accurate and robust.

4.3. Knowledge Conceptual Drift

In this subsection we provide systematic way to evaluate and propagate knowledge conceptual drift. Technical conceptual drift at the metamodel layer often triggers subsequent knowledge conceptual drift, stemming from the continuous inference and transformation that the metamodel experiences throughout its evolutionary trajectory. Such shifts may necessitate a careful realignment of the metamodel with ontology to preserve semantic coherence. The need for realignment depend on how big and complex the metamodel changes are, small tweaks need little effort, but major changes require thorough redesign.

4.3.1. Drift Measurement: After propagation of changes at metamodel layer, in this subsection we provide details how we evaluate knowledge conceptual drift between metamodel and ontology layer.

Drift Case 1 - Controller Load Level: We evaluate knowledge conceptual drift, when a new functionality to a controller is added, with respect to each domain ontology class by employing the metric defined in Equation 6, enabling a granular, class-level assessment of semantic divergence over time.

$$\begin{aligned} \mathbf{sim}_{\mathcal{H}(\text{old})} &= \mathbf{score}(\mathcal{H}_{\text{old (C)}}, \mathcal{O}_{\text{C}}) \\ \mathbf{sim}_{\mathcal{H}(\text{new})} &= \mathbf{score}(\mathcal{H}_{\text{new (C)}}, \mathcal{O}_{\text{C}}) \\ \text{diff} &= |\mathbf{sim}_{\mathcal{H}(\text{old})} - \mathbf{sim}_{\mathcal{H}(\text{new})}| \end{aligned} \tag{6}$$

We leverage the **score** methods from the alignment process presented in (Abbasi et al. 2026), which enables independent monitoring of both semantic and structural variations of metamodel entities with respect to ontology concepts. This allows for precise quantification of the drift's nature, distinguishing whether it stems from changes in the class structure or purely from semantic shifts. This capability is made possible by the metamodel-ontology matching method (`SSM-OM`) proposed in (Abbasi et al. 2026), which tracks structural and semantic features of the metamodel to align it with the domain ontology. Then we compare the previous $\mathcal{H}_{\text{old(C)}}$ and new metamodel entities $\mathcal{H}_{\text{new(C)}}$ to quantify the degree of change. We further introduce an alternative metric to quantify knowledge conceptual drift that accounts for the scale or granularity of the ontology. This measure, formally presented in Equation 7.

$$\text{Drift Score}(c) = \frac{1}{|\mathcal{O}|} \sum_{o \in \mathcal{O}} |\text{sim}_{\mathcal{H}(\text{old})} - \text{sim}_{\mathcal{H}(\text{new})}| \tag{7}$$

Where $|\mathcal{O}|$ is the number of classes in an ontology. This measure enables a more nuanced evaluation by incorporating the influence of ontology size, thus providing a deeper understanding of how the extent of the ontology affects conceptual alignment shifts over time or between models. Equations 6 and 7 quantify the magnitude of structural and semantic changes in the metamodel. Based on these drift scores, the DT maintainer can determine whether the metamodel should be realigned with the ontology to preserve equivalence correspondences. We adapted this from *SemaDrift framework*, which measures semantic drift across ontology versions (Stavropoulos et al. 2017, 2016) and integrates textual and graph-based metrics to detect and quantify conceptual changes over time.

Drift Case 2 - Pressure Sensor: In this scenario, we only check whether a sensor concept exists in the ontology. If so, it is (re)aligned; if a similar concept exists, it is aligned to the closest match. Otherwise, the ontology and DT are extended, causing a structural alignment change due to new sensing capabilities as illustrated in Example 4.1.

Example 4.1 *Consider* `Controller` *entity, which already includes sensors such as* `Proximity`, `TemperatureSensor`, *and* `HumiditySensor`. *When a new* `PressureSensor` *is installed on the AT, the model and metamodel layers are first updated as described in sections 4.1 and 4.2. If the ontology contains a concept such as* `EnvironmentalSensor` *or* `PressureSensor`, *the new sensor is aligned accordingly. Otherwise, ontology must be extended by introducing a new concept (e.g.,* `PressureSensor`*) or another ontology must be selected with right concepts, followed by updation of DT subject.*

4.3.2. Drift Propagation: After evaluating knowledge conceptual drift for each case in section 3.1, the metamodel and domain ontology must be realigned to preserve the DT's semantic fidelity, with any metamodel updates propagated to the ontology to maintain semantic grounding.

We provide results of our case-based generalisation approach in Table 2, where we only focus on structural adaptations significant for managing semantic drift systematically. Table 2 summaries semantic drift management in an air quality use case, covering two drift cases in section 3. We also provide workflow of semantic drift management in Figure 3. For each case, vertical drift variants are considered: (i) structural drift, (ii) technical conceptual drift, and (iii) knowledge conceptual drift. Structural drift is mainly detected through entity recognition applied to data and leads to inconsistencies and misalignment across higher abstraction layers. In this work, identification is performed exclusively for structural drift emerging data. Evaluation methods include drift scoring (Equation 2, 3 and 7) as well as entity and concept validation. Finally, the identified drifts are addressed through various propagation actions for correction, such as DT entity re-instantiation, entity inference, entity instantiation, concept alignment and realignment. Semantic drift becomes relevant when the ontology layer is examined at a higher level of abstraction, whereas at lower levels it is perceived merely as syntactic drift.

5. Discussion

This section qualitatively discusses the proposed semantic drift management approach for multi-layered, model-driven DTs, introducing a three-step process: (i) identification, (ii) evaluation, and (iii) propagation, across four abstraction layers.

Semantic Drift Cases - Air Quality Usecase	Semantic Drift Variant	Identification	Evaluation	Propagation
Drift Case 1 - Load Level	Structural Drift	Entity Recognition	Drift Score - Equation 2	DT Entity Re-instantiation
	Technical Conceptual Drift		Drift Score - Equation 3	Entity Inference
	Knowledge Conceptual Drift		Drift Score - Equation 7	Concept Re-alignment
Drift Case 2 - Pressure Sensor	Structural Drift	Entity Recognition	Drift Score - Equation 2	DT Entity Instantiation
	Technical Conceptual Drift		Entity Validation	Entity Inference
	Knowledge Conceptual Drift		Concept Validation	Concept Alignment

Table 2 Case-based Generalisation Results

5.1. Automation

We provide a semi-automated drift management strategy, where structural drift is detected automatically from incoming data, but decisions regarding evaluation and propagation of changes across the model, metamodel, and ontology layers remain under the supervision of the DT maintainer. This human-in-the-loop design prevents uncontrolled DT evolution, diverging away from DT's intended subject, purpose or goal and aligns with the broader vision of progressively adaptive and autonomous DT architectures discussed (Hribernik et al. 2021). Full autonomy is therefore not considered but envisioned as future work. To transition from manual maintainer validation to fully autonomous drift correction, a gradual and controlled automation strategy can be adopted, consistent with online adaptation concepts, some aspects discussed as follows:

1. **Confidence-based Validation:** These mechanisms can be introduced, where each detected structural drift instance is assigned a confidence score based on data consistency, historical adaptation patterns, and rule compliance. Such risk-aware automation aligns with adaptive DT governance principles reported in (Listl et al. 2024). Low-risk, high-confidence changes (e.g., integration of a known sensor type with predefined semantics) may be automatically approved, while high-impact modifications remain under human supervision. Over time, as validation models mature, the proportion of automated decisions can increase.

2. **Governance Rules:** Policy-driven governance rules can constrain autonomous adaptations by formally encoding the DT's subject, purpose, and operational boundaries. This policy-based control layer ensures that structural and semantic modifications remain within predefined safety and functional limits, an approach consistent with adaptive DT frameworks proposed in (Hribernik et al. 2021).

3. **Reinforcement Learning:** This can be incorporated to optimize drift correction decisions. Reinforcement learning based DT control frameworks proposed in (Berg et al. 2025) demonstrate how learning agents can iteratively improve decision policies based on performance feedback. In the context of drift management, an reinforcement learning agent could learn from historical maintainer approvals, rejections, and system outcomes, optimizing correction strategies while balancing structural consistency, behavioral accuracy, and operational stability.

4. **Outcome-based Validation:** Continuous outcome validation loops, as emphasized in (Nikula et al. 2020), can support safe autonomy. After applying a drift adjustment, the DT monitors system behavior and compares predicted versus actual outcomes. If deviations increase, rollback mechanisms automatically revert changes, creating a self-correcting feedback loop that mitigates the risk of uncontrolled evolution.

By integrating confidence-based validation, governance policies, reinforcement learning, and continuous outcome validation loops, DT manual maintainer validation can progressively evolve into safe and scalable autonomous drift correction, while preserving the DT's intended objectives and operational integrity.

5.2. Generalizability

Air quality management usecase discussed in section 3, explains, in a practical way, how different types of semantic drift can appear and how they are handled. It illustrates a complete workflow for detecting and managing changes over time. It shows how different semantic drift variants may occur, for instance, when new functionalities or components are introduced at AT. The usecase also describes the various situations that can arise while addressing semantic drift. These include identifying that a semantic drift has occurred, analyzing its impact on the DT, deciding on the appropriate response, and updating the abstraction layers inside DT to reflect the new understanding. Although the workflow is presented in the context of air quality management usecase, it is quite general and not limited to this specific domain. The same step-by-step process can be applied to other use cases where concepts, terminology, or data interpretations evolve over time.

The proposed workflow is general and independent of any specific tool. Techniques such entity recognition, syntactic similarity matching or metamodel inference are implementation options, not restrictions of the method itself. In our current implementation, entity recognition serves as an example instantiation of the drift identification step. While entity recognition itself is not the core contribution, it is essential for detecting changes, such as new components or functionalities, added on AT from data, ensuring that the DT remains aligned with its intended subject and scope. Likewise, we use lightweight similarity measures (e.g., Jaccard similarity and the Ratcliff algorithm) to keep the process simple and interpretable. However, these components can be replaced with more advanced entity

recognition process, semantic matching or ontology alignment approaches without changing the overall workflow.

5.3. Structural Adaptability

In our tool-independent, semi-automated workflow for semantic drift management, we employ JSMF to support metamodel inference (Sottet & Biri 2016). Within this data-driven modeling context, JSMF enables metamodel evolution patterns and supports essential `CRUD (Create, Read, Update, Delete)` operations, facilitating flexible structural adaptation as new data emerges. When applying metamodel inference (i.e., automatically inferring structural definitions from models or data), JSMF supports following metamodel evolution patterns:

1. **Incremental Extension:** new classes, attributes, and relationships can be incrementally added to the metamodel as new data patterns and models emerge, reflecting JSMF flexible and data-driven approach to metamodel evolution.

2. **Merge and Split:** JSMF supports both merge and split evolution patterns in a data-driven metamodel context, as illustrated in the air quality use case (see Figure 1). The air quality exemplar metamodel already contains different sensor types under a controller (e.g., `TemperatureSensor`, `HumiditySensor`, and `Proximity`). If these sensors share similar structural properties such as value and unit, they can be merged into a more generalized superclass (e.g., `EnvironmentalSensor`) to reduce redundancy and improve abstraction. Conversely, when new data introduces a distinct sensor type, such as the newly added `PressureSensor` in **Drift Case 2** (see section 3.1.2), the metamodel can be extended or the generic sensor structure can be split into more specialized subclasses to reflect emerging functional differences. This capability enables the metamodel to adapt dynamically as new sensor types, properties, or behaviors appear in the evolving air quality monitoring system.

3. **Conformance Relaxation:** JSMF supports partial conformance and flexible metamodel constraints, enabling the structure to adapt without enforcing strict static rules upfront.

When supporting metamodel inference, typical `CRUD (Create, Read, Update, Delete)` operations are performed on the metamodel. This makes JSMF suitable for model-driven adaptive systems, enabling continuous alignment and synchronization between the AT and its DT as models evolve.

5.4. Behavioural Adaptability

In this work, we address vertical drift, which emerges from the structural changes in data. Vertical drift mostly impacts structural models, i.e., static, which only describe the static aspect of DT and its AT. We also acknowledge that behavioral drift can occur when DT is operational and structural adaptation is a necessary precursor to behavioral evolution. Behavioral drift impacts the behavioral models and occurs when actions or real-world outcomes change over time even though the underlying rules of the behavioral model in DT remain unchanged, due to shifts in system dynamics or the surrounding environment. It typically occurs when the DT's behavioral models are inconsistent with the system it represents. This systematic semantic drift management approach can be extended and strengthened in future by ensuring that all layers of DT remain semantically aligned and actions taken by AT are continuously validated against behavioral models existing in DT. Structural updates always come first. For example, if a new `PressureSensor` is added to AT, DT must update its structure at all abstraction levels, including the model, metamodel, and ontology, so the new component is properly represented and understood across the system. Once these structural changes are aligned, the behavioral models and their metamodel must also be updated. Only after both structural and behavioral updates are complete, behaviors, such as ventilation control or alarm activation, can operate correctly. By maintaining consistency across structural layers and continuously comparing expected behavior outcomes with real-world results, the DT can detect deviations early and adapt safely. This combination of cross-layer semantic alignment and ongoing action validation provides a reliable foundation for managing behavioral changes without introducing errors or unsafe system responses.

6. Conclusion

DTs replicate real-world behavior through heterogeneous models in multi-layered paradigm, enhancing systems-of-systems with insights and optimizations. However, the dynamic nature of real entities can cause semantic drift, which must be addressed proactively to maintain DT effectiveness. Our research contributes to manage semantic drift systematically in multi-layered, model-driven DTs, especially **vertical drift**, in **bottom-up** manner, supporting only **structural adaptations** across data, model, metamodel and ontology layers. Our systematic approach revolves around two core concepts, i.e., flexibility and alignment, which ensures that metamodel is refined, extended, aligned and continuously updated in response to real world evolution. We provide methods to identify, measure/evaluate and propagate different variants of semantic drift exploiting feature of heterogeneous model alignment approach given in (Abbasi et al. 2026), finally presenting the results using case-based generalisation strategy for air quality usecase.

Looking ahead, we aim to extend this systematic approach to address horizontal drift and support behavioural adaptation across heterogeneous models, explicitly accounting for the fact that structural changes inherently lead to behavioural changes. We also plan to evaluate the approach using additional use cases to achieve stronger generalisation.

Acknowledgments

Supported by Luxembourg National Research Fund (FNR), project MDDT-SD grant number C22/IS/17153694.

References

Abbasi, F., Brimont, P., Pruski, C., & Sottet, J.-S. (2024). Understanding semantic drift in model driven digital twins. In *Proceedings of the acm/ieee 27th international conference*

on model driven engineering languages and systems (pp. 419–430).

Abbasi, F., Muzammal, M., Qu, Q., Riaz, F., & Ashraf, J. (2024). Snca: Semi-supervised node classification for evolving large attributed graphs. *Big Data Mining and Analytics*, *7*(3), 794–808.

Abbasi, F., Muzammal, M., Qureshi, K. N., Javed, I. T., Margaria, T., & Crespi, N. (2022). Exploiting optimised communities in directed weighted graphs for link prediction. *Online Social Networks and Media*, *31*, 100222.

Abbasi, F., Pruski, C., & Sottet, J.-S. (2025). Semantic drift evaluation in language and data-specific digital twin frameworks. *Future Generation Computer Systems*, 108240.

Abbasi, F., Sottet, J.-S., & Pruski, C. (2026). *Semantic grounding of digital twin metamodels using rdf graphs.* Retrieved from https://arxiv.org/abs/2512.15281

Abbasi, F., Talat, R., & Muzammal, M. (2019). An ensemble framework for link prediction in signed graph. In *2019 22nd international multitopic conference (inmic)* (pp. 1–6).

Alskaif, T., Babur, Ö., Bordeleau, F., Cleophas, L., Combemale, B., Denil, J., ... others (2025). Evolution at the core of digital twin engineering.

Anik, S. M. H., Gao, X., & Meng, N. (2025). A comprehensive indoor environment dataset from single-family houses in the us. *Data*, *10*(3), 35.

Barricelli, B. R., Casiraghi, E., & Fogli, D. (2019). A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE access*, *7*, 167653–167671.

Berg, H. S., Menges, D., Tengesdal, T., & Rasheed, A. (2025). Digital twin syncing for autonomous surface vessels using reinforcement learning and nonlinear model predictive control. *Scientific Reports*, *15*(1), 9344.

Bordeleau, F., Combemale, B., Eramo, R., Van Den Brand, M., & Wimmer, M. (2020). Towards model-driven digital twin engineering: Current opportunities and future challenges. In *International conference on systems modelling and management* (pp. 43–54).

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-driven software engineering in practice*. Morgan & Claypool Publishers.

Chevallier, Z., Finance, B., & Boulakia, B. C. (2020). A reference architecture for smart building digital twin. *SeDiT@ ESWC*, *2020*(5).

Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., & Wortmann, A. (2022). A cross-domain systematic mapping study on software engineering for digital twins. *Journal of Systems and Software*, *193*, 111361.

Dittler, D., Lierhammer, P., Braun, D., Müller, T., Jazdi, N., & Weyrich, M. (2022). An agent-based realisation for a continuous model adaption approach in intelligent digital twins. *arXiv preprint arXiv:2212.03681*.

Eramo, R., Bordeleau, F., Combemale, B., van Den Brand, M., Wimmer, M., & Wortmann, A. (2021). Conceptualizing digital twins. *IEEE Software*, *39*(2), 39–46.

Etien, A., & Anquetil, N. (2024). Automatic recommendations for evolving relational databases schema. *arXiv preprint arXiv:2404.08525*.

Fuller, A., Fan, Z., Day, C., & Barlow, C. (2020). Digital twin: enabling technologies, challenges and open research. *IEEE access*, *8*, 108952–108971.

Garcés, K., Jouault, F., Cointe, P., & Bézivin, J. (2009). Managing model adaptation by precise detection of metamodel changes. In *Model driven architecture-foundations and applications: 5th european conference, ecmda-fa 2009, enschede, the netherlands, june 23-26, 2009. proceedings 5* (pp. 34–49).

Govindasamy, H. S., Jayaraman, R., Taspinar, B., Lehner, D., & Wimmer, M. (n.d.). Air quality management: An exemplar for model-driven digital twin engineering. in 2021 acm. In *Ieee international conference on model driven engineering languages and systems companion (models-c)* (pp. 229–232).

Harel, D., & Rumpe, B. (2004). Meaningful modeling: what's the semantics of" semantics"? *Computer*, *37*(10), 64–72.

Hellwig, A., Michael, J., Pfeiffer, J., Rumpe, B., Thillmann, H., & Wortmann, A. (2025). Digital twins in manufacturing and the use of models.

Hribernik, K., Cabri, G., Mandreoli, F., & Mentzas, G. (2021). Autonomous, context-aware, adaptive digital twins—state of the art and roadmap. *Computers in Industry*, *133*, 103508.

Jeong, D.-Y., Baek, M.-S., Lim, T.-B., Kim, Y.-W., Kim, S.-H., Lee, Y.-T., ... Lee, I.-B. (2022). Digital twin: Technology evolution stages and implementation layers with technology elements. *Ieee Access*, *10*, 52609–52620.

Kannapinn, M., Schäfer, M., & Weeger, O. (2024). Twinlab: a framework for data-efficient training of non-intrusive reduced-order models for digital twins. *Engineering Computations*.

Lehner, D., Garmendia, A., & Wimmer, M. (2021). Towards flexible evolution of digital twins with fluent apis. In *2021 26th ieee international conference on emerging technologies and factory automation (etfa)* (pp. 1–4).

Lehner, D., Gil, S., Mikkelsen, P. H., Larsen, P. G., & Wimmer, M. (2023). An architectural extension for digital twin platforms to leverage behavioral modelsbehaviors. In *2023 ieee 19th international conference on automation science and engineering (case)* (pp. 1–8).

Lehner, D., Zhang, J., Pfeiffer, J., Sint, S., Splettstößer, A.-K., Wimmer, M., & Wortmann, A. (2025). Model-driven engineering for digital twins: a systematic mapping study. *Software and Systems Modeling*.

Listl, F. G., Dittler, D., Hildebrandt, G., Stegmaier, V., Jazdi, N., & Weyrich, M. (2024). Knowledge graphs in the digital twin: A systematic literature review about the combination of semantic technologies and simulation in industrial automation. *IEEE Access*, *12*, 187828–187843.

Liu, J., Ji, Q., Zhou, H., Du, C., Liu, X., & Li, M. (2024). A multi-dimensional evolution modeling method for digital twin process model. *Robotics and Computer-Integrated Manufacturing*, *86*, 102667.

Lu, R., Rausch, C., Bolpagni, M., Brilakis, I., & Haas, C. T. (2020). Geometric accuracy of digital twins for structural health monitoring. In *Structural integrity and failure.* IntechOpen London, UK.

Mertens, J., & Denil, J. (2023). Digital-twin co-evolution

using continuous validation. In *Proceedings of the acm/ieee 14th international conference on cyber-physical systems (with cps-iot week 2023)* (pp. 266–267).
Mertens, J., Klikovits, S., Bordeleau, F., Denil, J., & Haugen, Ø. (2024). Continuous evolution of digital twins using the dartwin notation. *Software and Systems Modeling*, 1–22.
Michael, J., Cleophas, L., Zschaler, S., Clark, T., Combemale, B., Godfrey, T., … others (2025). Model-driven engineering for digital twins: Opportunities and challenges. *Systems Engineering*.
Michael, J., David, I., & Bork, D. (2024). Digital twin evolution for sustainable smart ecosystems. In *Proceedings of the acm/ieee 27th international conference on model driven engineering languages and systems* (pp. 1061–1065).
Muctadir, H. M., Cleophas, L., & van den Brand, M. (2024). Maintaining consistency of digital twin models: exploring the potential of graph-based approaches. In *2024 50th euromicro conference on software engineering and advanced applications (seaa)* (pp. 152–159).
Muctadir, H. M., Kamburjan, E., Cleophas, L., & van den Brand, M. (2025). A consistency management framework for digital twin models. *Available at SSRN 5105174*.
Muzammal, M., Abbasi, F., Qu, Q., Talat, R., & Fan, J. (2020). A decentralised approach for link inference in large signed graphs. *Future Generation Computer Systems*, *102*, 827–837.
Nikula, R.-P., Paavola, M., Ruusunen, M., & Keski-Rahkonen, J. (2020). Towards online adaptation of digital twins. *Open Engineering*, *10*(1), 776–783.
Pronost, G., Mayer, F., Camargo, M., & Dupont, L. (2024). Digital twins along the product lifecycle: A systematic literature review of applications in manufacturing: [version 2; peer review: 2 approved, 2 approved with reservations]. *Digital Twin*, *1*(2), 3.
Robles, J., Martín, C., & Díaz, M. (2023). Opentwins: An open-source framework for the development of next-gen compositional digital twins. *Computers in Industry*, *152*, 104007.
Sottet, J.-S., & Biri, N. (2016). Jsmf: a flexible javascript modelling framework. In *Workshop on flexible model driven eng.(flexmde)*.
Sottet, J.-S., Brimont, P., Pruski, C., & Abbasi, F. (2025). How to leverage digital twin for system design? In *Modelsward* (pp. 305–312).
Stavropoulos, T. G., Andreadis, S., Riga, M., Kontopoulos, E., Mitzias, P., & Kompatsiaris, I. (2016). A framework for measuring semantic drift in ontologies. In *Semantics (posters, demos, success)*.
Stavropoulos, T. G., Kontopoulos, E., Meroño-Peñuela, A., Tachos, S., Andreadis, S., & Kompatsiaris, Y. (2017). Cross-domain semantic drift measurement in ontologies using the semadrift tool and metrics. In *Mepdaw/ldq@ eswc* (pp. 59–72).
Sun, W., Lei, S., Wang, L., Liu, Z., & Zhang, Y. (2020). Adaptive federated learning and digital twin for industrial internet of things. *IEEE Transactions on Industrial Informatics*, *17*(8), 5605–5614.
Thelen, A., Zhang, X., Fink, O., Lu, Y., Ghosh, S., Youn, B. D., … Hu, Z. (2022). A comprehensive review of digital twin—part 1: modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization*, *65*(12), 354.
Tong, X., Bao, J., & Tao, F. (2024). Co-evolutionary digital twins: A multidimensional dynamic approach to digital engineering. *Advanced Engineering Informatics*, *61*, 102554.
Tzachor, A., Sabri, S., Richards, C. E., Rajabifard, A., & Acuto, M. (2022). Potential and limitations of digital twins to achieve the sustainable development goals. *Nature Sustainability*, *5*(10), 822–829.
VanDerHorn, E., & Mahadevan, S. (2021). Digital twin: Generalization, characterization and implementation. *Decision support systems*, *145*, 113524.
Whittle, J., Hutchinson, J., & Rouncefield, M. (2013). The state of practice in model-driven engineering. *IEEE software*, *31*(3), 79–85.
Wieringa, R., & Daneva, M. (2015). Six strategies for generalizing software engineering theories. *Science of computer programming*, *101*, 136–152.
Zhang, H., Qi, Q., & Tao, F. (2022). A consistency evaluation method for digital twin models. *Journal of Manufacturing Systems*, *65*, 158–168.

About the authors

Faima Abbasi is a Ph.D. student at Luxembourg Institute of Science and Technology (LIST) and University of Luxembourg. Her research interest lie in intersection of digital twins, semantic web, graph theory, data mining and network science. You can contact the author at faima.abbasi@list.lu.

Jean-Sébastien Sottet is researcher at the Luxembourg Institute of Science and Technology (LIST). His research interest is related to model-engineering ranging from design of systems to perception of model and cognitive process of modelling. He has led both academic and industry collaborations, applying model-driven approaches to diverse fields such as governance, risk and regulatory compliance, enterprise architecture, software optimization, and, most recently, model-driven digital twins for energy transition, transportation and advanced network system. Jean-Sébastien's work aims to bridge theory and practice, driving innovation through model engineering: harmonizing people's concerns and views, disciplines, technologies, and data, from informal to executable representations. You can contact the author at jean-sebastien.sottet@list.lu.

Cedric Pruski is researcher at the Luxembourg Institute of Science and Technology (LIST). His research interests are artificial intelligence and knowledge representation and reasoning. He received an "Habilitation à Diriger des Recherches" from university Paris-Saclay and a PhD in computer science from both university of Paris-Sud and university of Luxembourg. He is the co-author of more than 100 scientific articles. He coordinated national and international research projects that have generated many publications in major conferences, patents, peer-reviewed journals. You can contact the author at cedric.pruski@list.lu.