Future Generation Computer Systems 177 (2026) 108240

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Semantic drift evaluation in language and data-specific digital twin frameworks

Faima Abbasi [a,b,*], Cédric Pruski [a], Jean-Sébastien Sottet [a]

[a] Luxembourg Institute of Science and Technology, 5 Avenue des Hauts-Fourneaux, Esch-sur-Alzette, L-4362, Luxembourg
[b] FSTM, University of Luxembourg, 2 Av de l'Universite, Esch-sur-Alzette, L-4365, Luxembourg

ARTICLE INFO

Editor: Prof. Michela Taufer

Keywords:
Digital twins
Semantic drift
Digital models

ABSTRACT

Digital Twin (DT) technology is a key enabler of Industry 4.0, integrating diverse digital models to optimize processes, enhance decisions, and support predictive maintenance. However, as physical systems evolve, delays in synchronization can cause semantic drift, leading to discrepancies between digital and real-world entities. Effective semantic drift management needs DT frameworks that support all modeling layers, i.e., data, model, metamodel, and ontology, and provide different model management mechanisms. In this paper, we make three key contributions. First, we define a comprehensive set of requirements to address semantic drift effectively, drawing on both our understanding of the phenomenon and a real-world use case on mobility. These requirements capture the essential characteristics needed to maintain synchronization between the digital and physical domains as systems evolve. Second, we introduce a set of metrics designed to evaluate the ability of DT platforms to tackle semantic drift, grounded in both the defined requirements and insights of the use-case. Finally, we apply these metrics to evaluate four existing DT platforms demonstrating their utility for identifying the limitations of current frameworks in handling semantic drift. Through this evaluation, we highlight the strengths and weaknesses of these platforms, providing a foundation for future improvements in DT infrastructure.

1. Introduction

The DT concept, central to Industry 4.0, represents a dynamic interface between physical and virtual entities, encompassing all product attributes and behaviors [1]. A fundamental three-dimensional DT application consists of three constructs: (i) physical element, (ii) virtual element, and (iii) bi-directional data flow. The virtual element integrates heterogeneous models, including (i) physics-based, (ii) geometric, (iii) behavioral, and (iv) machine-learning models. Recent advancements extend this model by incorporating DT data and services [2]. DT processes multi-temporal, multi-dimensional, multi-source, and heterogeneous data. Services, under the everything-as-a-service (XaaS) model, are central to DT, enabling applications like simulation, monitoring, optimization, and health management, while relying on third-party services for data, algorithms, and ongoing platform support for software and model development. DTs are crucial throughout a system's life cycle, enabling optimization, monitoring, and diagnostics. However, integrating heterogeneous digital models from diverse stakeholders is challenging due to differences in semantics, schema, and syntax. With internet of things (IoT), big data, and cloud computing, DTs enhance industrial and scientific systems [2]. DTs harness semantic technologies, i.e., ontologies, knowledge graphs, natural language processing, and semantic web standards, to ensure interoperability and unified system representation. However, they are prone to semantic drift, arising from evolving environments, data inaccuracies, and modeling limitations, disrupting alignment between virtual models and the real world. Semantic drift has been observed in various fields: (i) machine learning [3], (ii) software architecture [4], and (iii) semantic web [5] and (iv) model-driven engineering [6], but has yet to be extensively explored in DTs. Addressing semantic drift in DT involves: (i) identifying changes, (ii) characterizing their nature, (iii) maintaining consistency across digital models, and (iv) propagating updates across digital models and interconnected components. Robust DT frameworks must effectively manage these phases across different modelling layers, i,e., data, models, metamodels and ontologies, to sustain system and DT integrity. This can be done through model management mechanisms such as migration, versioning, evolution and traceability. Collaboration between DT frameworks and modelling experts, managing vertical and horizontal inconsistencies, and securing data are key to manage semantic drift in DTs [7].

This work explores semantic drift in model-driven DTs, considering diverse models, metamodels, and ontologies across modeling layers. We introduce a mobility data model for DT setup and evaluate language and

* Corresponding author.
E-mail addresses: faima.abbasi@list.lu (F. Abbasi), cedric.pruski@list.lu (C. Pruski), jean-sebastien.sottet@list.lu (J.-S. Sottet).

https://doi.org/10.1016/j.future.2025.108240
Received 9 January 2025; Received in revised form 30 October 2025; Accepted 1 November 2025
Available online 8 November 2025

data-specific frameworks based on their capacity to *identify, characterize, and provide any kind of semantic drift support* through experiments in diverse scenarios. We focus on answering the following research questions:

RQ1 What are essential requirements for DT frameworks to address and manage semantic drift?
RQ2 What metrics are used to evaluate the capability of DT frameworks for addressing and managing semantic drift?
RQ3 How do language and data-specific DT frameworks offer robust support for addressing and managing semantic drift?

We summarize our contributions as follows:

1. We elicit key requirements for *identifying, characterizing, and managing semantic drift*, based on a detailed analysis of the phenomenon and a practical mobility use case with prototypical scenarios.
2. We propose robust metrics, grounded in elicited requirements, to evaluate DT frameworks' ability to address semantic drift.
3. We employ these metrics to rigorously assess language and data-specific DT frameworks, highlighting their limitations in managing semantic drift.

This paper covers: (i) related work in Section 2, (ii) semantic drift and preliminaries in Section 3, (iii) requirements and a mobility use case in Section 4, and (iv) DT framework evaluation metrics, setup, and validation in Sections 5–7, followed by discussion and conclusion in Sections 8 and 9.

2. Related work

This section explores how incorporating linked data into model-driven engineering (MDE) mitigates semantic drift in DTs. We highlight key research contributions in DT evolution and maintenance across three research areas: (i) data management, (ii) model-driven concepts and (iii) linked data technologies.

2.1. Data management

Integrating IoT with DTs enables real-time monitoring, predictive maintenance, and smarter decisions. Effective data management ensures DTs stay synchronized with physical systems through continuous IoT streams. Key aspects of data management include [8]: (i) integration of diverse data sources, (ii) quality assurance, (iii) scalability, and (iv) security/privacy. Despite these advances, a major challenge, i.e., data drift, persist, which involves changes in data properties that reduce model accuracy and DT reliability. In IoT systems, data drift occurs due to sensor degradation, environmental changes, or operations, undermines DT accuracy. Classification of data drift is given in [9], while formal definitions are provided in [7]. Recent studies propose advanced methods to detect and mitigate some variants of drift in DTs. Armijo et al. [10] use IoT-enabled monitoring and machine learning to detect and mitigate structural drift and adapt DTs in real time, enhancing health diagnostics. Hasan et al. [11] introduce a GAN-based feature drift detector for sensor data. For label drift, Tian et al. [12] propose a continuous learning approach, while Wu et al. [13] address concept drift using a dynamic ensembling framework. Together, these approaches improve DT accuracy, fault detection, and adaptability in evolving environments.

2.2. Model-driven concepts

MDE provides a structured foundation for DT evolution and maintenance. This subsection outlines key concepts: (i) model management, (ii) consistency management, and (iii) modeling paradigms, to address semantic drift in model-driven DTs.

2.2.1. Model management

In DT ecosystems, effective model management ensures synchronization and integrity between DTs and their physical counterparts. DT rely on models that mirror their physical counterparts, anchored by metamodels defining their structure and semantics. As DTs evolve, coordinated model and metamodel co-evolution is essential. Effective model management ensures seamless migration, versioning, traceability, and automated runtime updates, enabling adaptive and reliable DT maintenance [14]. Studies use (i) machine learning [15], (ii) simulations [15,16], and (iii) federated learning [17] to emphasize the need for DT model management and evolution, ensuring adaptability, real-time accuracy, and autonomous decisions. *TwinLab* integrates ML and simulation to enable fast, accurate ROMs [15]; *DarTwin* applies MDE for modular DT evolution [16]; and Sun et al. address scalable model training via federated learning [17]. Metamodels provide the structural and semantic backbone of domain models, evolving iteratively to ensure accuracy and adaptability. However, metamodel evolution in model-driven DTs remains underexplored. Lehner et al. [18] address this by introducing a hierarchical model for consistent schema propagation across types, properties, and instances. Their fluent API enables runtime modifications, integrated into a model management framework with an evolution engine supporting migration, versioning, and data handling, enhancing automation and preserving data integrity throughout the DT lifecycle.

2.2.2. Consistency management

Consistency management ensures alignment, correctness, and coherence between DT models, instances, and their physical counterparts as they evolve. However, few studies offer robust solutions to address challenges like data drift, latency, and structural change. Muctadir et al. [19] propose a graph-based framework utilizing Neo4j and Cypher to manage inter-model consistency through unified data storage and rule-based validation. Validated on autonomous truck docking and Xtext grammar case studies, the approach effectively addresses heterogeneity and dynamic model evolution. In a complementary study, Muctadir et al. [20] further extend this by ensuring coherence among heterogeneous models at the same abstraction level. Their method, grounded in MDE principles, leverages structured model relations and automated validation to unify DTs and MDE, enhancing cross-domain synchronization and consistency management.

2.2.3. Modeling paradigms

Advanced modeling paradigms, i.e., (i) multi-level, (ii) multi-view, and (iii) mega-modeling, provide scalable, structured frameworks to represent complex systems across varying abstractions and semantic dimensions. Multi-level modeling structures systems across hierarchical abstraction layers, enabling scalable and semantically rich DT evolution [21]. Liu et al. [22] address proposed an evolutionary concurrent modeling method that captures the semantic, temporal, and feature-level dynamics of DT processes. Their multi-layered structure allows independent yet synchronized evolution, ensuring consistency and adaptability across layers. This promotes efficient co-evolution and real-time responsiveness. Multi-view modeling creates distinct views focusing on specific system aspects, exemplified by *HoloWoT*, which integrates mixed reality and IoT for dynamic, context-aware DT interfaces [23]. A mega-modelling paradigm offers an integrated, multi-domain view of a system, unifying diverse models, abstraction levels, and dependencies into a cohesive, synchronized structure. Bucaioni et al. [24] present an MDE framework for federated DTs in industrial systems, focusing on integration, validation, and lifecycle continuity. Combining multi-view modeling, model weaving, and mega-modeling, it ensures traceability and interoperability between digital models and shadows. Using UML/SysML and domain specific languages (DSL), the framework supports automated validation, prediction, and adaptive evolution.

Table 1
Nomenclature.

Notation	Description
[0,t], [t+1,t+m]	Time interval, where $\{t \mid t \in \mathbb{N}\}$
$\mathcal{S}$	System or Physical Element
$\mathcal{T}$	Twin or Virtual Element
$\Delta\mu$, $\Delta\eta$	Semantic Drifts
Φ	Semantic Drift Threshold
$\mathcal{I}$, $\mathcal{H}$, $\mathcal{O}$	Model, Metamodel, Ontology
C, E, P, R, I, A	Concept, Entity, Property, Relation, Individual & Axiom
$\mathcal{S}_{[0,t]} \overset{\mu}{\Leftrightarrow} \mathcal{T}_{[0,t]}$	Real-world mapping to Virtual Element
$\mathcal{I}_{[0,t]} \overset{\eta}{\Leftrightarrow} \mathcal{H}_{[0,t]}$	Mapping Models layer to metamodel layer
$\mathcal{I}_{[0,t]}, \mathcal{H}_{[0,t]} \overset{\eta}{\Leftrightarrow} \mathcal{O}_{[0,t]}$	Mapping Model and metamodel layer to Ontology layer

2.3. Linked data technologies

Linked data technologies, based on standards like resource description framework (RDF) and universal resource identifier (URI), enable seamless publication and linking of structured, machine-readable data across distributed systems. Serving as the foundation for ontologies and knowledge graphs, they provide interoperable frameworks for organizing and retrieving complex DT information [25]. Ontologies define domain concepts, while knowledge graphs represent real-time and historical states [26–28]. Together, they enhance interoperability, traceability, and dynamic reasoning critical for evolving DT ecosystems. Ontologies provide a shared conceptual framework enabling semantic interoperability and structured domain representation [25].

Ontology management, continuous refinement to mirror domain evolution, is crucial yet underexplored in model-driven DTs [29]. Studies by Bao et al. [30] and Ren et al. [31] showcase ontology-driven synchronization and lifecycle governance to maintain accurate, context-aware DTs. Knowledge graphs, rooted in ontologies, represent interconnected entities and relationships to enable semantic search, contextual reasoning, and intelligent data integration [32,33]. In DTs, Eduard et al. [34] propose an automated self-adaptation framework using declarative lifecycle descriptions and a two-tier MAPE-K loop: one for behavioral adaptation, another for dynamic architectural restructuring. Leveraging semantic technologies and knowledge graphs, their approach detects lifecycle transitions and preserves architectural consistency without explicit transition modeling. Validated on a greenhouse DT, it outperforms traditional methods in scalability and efficiency.

Recent studies on DT evolution in MDE reveal gaps: isolated drift handling, lack of unified real-time frameworks, and limited focus on data quality and provenance. Federated architectures and automated adaptation pipelines remain underdeveloped, with co-evolution and linked data largely manual and weakly integrated.

3. Preliminaries

Semantic drift is defined as a *sudden, gradual, incremental, or recurring shift in a DT component's structure, precision, value, or meaning*, poses a critical threat to model coherence [7]. We formally define semantic drift in Definition 3.1 along some useful notations in Table 1. It undermines the alignment of heterogeneous digital models, including ontologies, software artifacts, and equations. Sustaining DT fidelity thus demands continual updates and rigorous semantic integration across modeling layers.

Definition 3.1. *(**Semantic Drift**) Suppose that a system $\mathcal{S}$ in the real world has a twin $\mathcal{T}$ with a set of heterogeneous digital models, metamodels and ontologies, at a time period [0,t]. $\mathcal{S}$ in the real world is connected to its $\mathcal{T}$ 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 $\mathcal{S}_{[0,t]} \overset{\mu}{\Leftrightarrow} \mathcal{T}_{[0,t]}$ changes to $\mathcal{S}_{[t+1,t+m]} \overset{\Delta\mu}{\Leftrightarrow} \mathcal{T}_{[t+1,t+m]}$, as the real world evolves.*

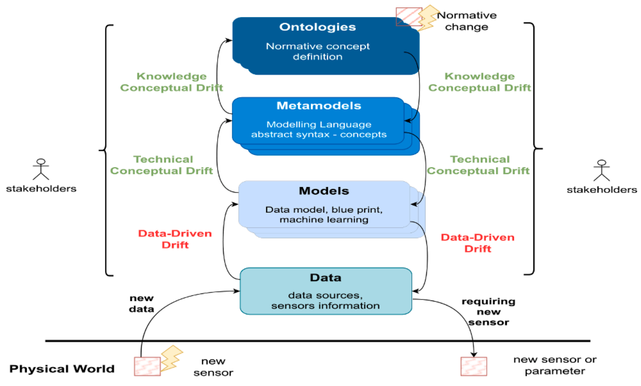

Normative change
Ontologies
Normative concept definition
Knowledge Conceptual Drift
Knowledge Conceptual Drift
Metamodels
Modelling Language abstract syntax - concepts
stakeholders
Technical Conceptual Drift
Technical Conceptual Drift
stakeholders
Models
Data model, blue print, machine learning
Data-Driven Drift
Data-Driven Drift
Data
data sources, sensors information
new data
requiring new sensor
Physical World
new sensor
new sensor or parameter

Fig. 1. Semantic drift in model-driven DT [7].

Note that, we use $\Delta\mu$ to demonstrate the semantic drift resulting from real-world evolution, making $\mathcal{S}$ inconsistent with $\mathcal{T}$. Since minor changes may not constitute drift, a threshold Φ determines when drift occurs.

A typical DT life cycle consists of [35]: (i) system design and creation phase, (ii) manufacturing or production phase, (iii) operational phase and (iv) end-of-life. Understanding semantic drift is a significant part of the DT life cycle and primarily emerges in operational phase. Furthermore, drift must be actively managed by DT maintainer in maintenance or evolution phase, a sub-phase overlapping operations, where the DT is updated, re-calibrated, and refined to preserve fidelity.

We consider four layers of modeling to cover different variants of drifts, as shown in Fig. 1. These modeling layers range from data collection up to the ontological layer. Four important constructs are illustrated by Fig. 1, 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. Data drift can take place at a lower level, where the system is represented in an abstract way, while the two upper layers, i.e., metamodels and ontologies, are susceptible to conceptual drift.

3.1. Data-driven drift

Data acts as an active link to connect the $\mathcal{S}$ and $\mathcal{T}$ of a DT and also plays a significant role at different levels, from low-level models up to the ontological description of the system. Data drift is a major cause of a cascading drift inside heterogeneous models, metamodels, and ontologies. More detailed examples of data-driven drift are given in [7].

Data influences the models to drift due to the direct connection with real-world data. Models are specific instances that represent abstract systems or concepts. Two different types of data drift have been studied in literature, i.e., (i) label, feature and concept [11,13,36], and (ii) structural drift [10,37]. Sensor aging, environmental change, or calibration changes can be potential reasons for the label, feature and concept drift in DT. This takes place when data captured by sensors changes from the threshold value for which they have been calibrated. A change in sensor calibration can even capture improper values, which no longer serve the intended purpose. Structural drift occurs when the operational characteristics of sensors are influenced while undergoing maintenance and repairs. This kind of activity can alter the unit of measurement or the entire structure of data captured by the sensor. The addition of new sensors in real-world settings can also be a cause of structural drift.

3.2. Technical conceptual drift

Structural drift is one of the potential reasons for technical conceptual drift. It occurs when models change arbitrarily over time, influencing change in metamodels. In the context of DT, $\mathcal{T}$ aims to simulate and predict the behavior of physical entities, due to the presence of diverse digital models and metamodels. We can define technical conceptual drift as follows:

Definition 3.2. *(**Technical Conceptual Drift**) Suppose that a twin $\mathcal{T}$ of a system $\mathcal{S}$ is composed of set of heterogeneous digital models $\mathcal{I}$ and metamodels $\mathcal{H} = (\mathbf{E}, \mathbf{P}, \mathbf{R})$, with a set of entities **E**, properties **P**, and relations **R**, conforming to the relation $\mathcal{I}_{[0,t]} \overset{\eta}{\Leftrightarrow} \mathcal{H}_{[0,t]}$. Technical conceptual drift occurs at the timestamp $[t+1]$ to time stamp $[t+m]$ (m can be any number), when $\mathcal{H}_{[0,1]} \neq \mathcal{H}_{[t+1,t+m]}$, due to a change in the underlying model $\mathcal{I}$ influenced by the evolving real-world environment, where the relation $\mathcal{I}_{[0,t]} \overset{\eta}{\Leftrightarrow} \mathcal{H}_{[0,t]}$ changes to $\mathcal{I}_{[t+1,t+m]} \overset{\Delta\eta}{\Leftrightarrow} \mathcal{H}_{[t+1,t+m]}$, no longer conforming to the real-world situation.*

Note that we use $\Delta\eta$ to define technical conceptual drift, between modeling layers, i.e., models and metamodels.

Heterogeneous digital models and metamodels are composed of different elements, which are susceptible to technical conceptual drift based on the concepts they represent. Some of the significant elements of heterogeneous digital models and metamodels include: (i) variables, parameters & constants, (ii) algorithms & rules, and (iii) relations & interactions. Variables, parameters & constants in heterogeneous digital models and metamodels are affected when new sensors are added to $\mathcal{S}$ or when the format of data is changed. This change is influenced by structural drift. Changes could also occur in *rules (constraints)* or *behavioral models (algorithms)* that are related to the metamodel and models.

This type of drift can make applications less effective in measuring and governing physical processes or system behavior. Changes can occur in the *direction, strength, and nature of the relationship* between different components of digital models and metamodels due to technical conceptual drift. This can lead to discrepancies in how elements are supposed to interact in DTs with evolving semantics. More detailed examples of technical conceptual drift are given in [7].

3.3. Knowledge conceptual drift

Technical conceptual drift can further propagate to the ontology layer. We can define ontology as *"formal, explicit, and shared description of concepts in a domain of discourse"*. We study drift in knowledge in the context of ontologies and formally illustrate in Definition 3.3.

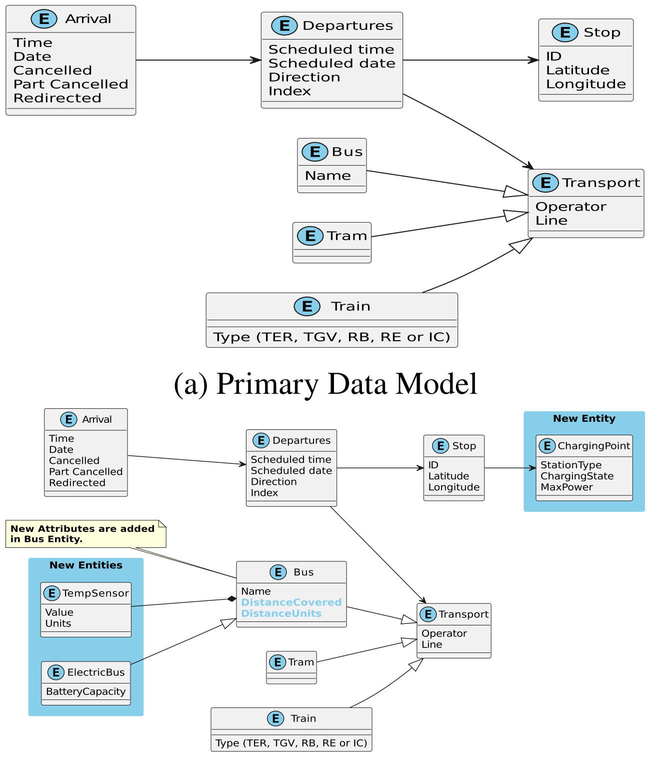
Definition 3.3. *(**Knowledge Conceptual Drift**) Suppose that a twin $\mathcal{T}$ of the system $\mathcal{S}$, at period [0,t], consists of a model $\mathcal{I}$ and metamodel $\mathcal{H}$ and an ontology $\mathcal{O} = (\mathbf{C}, \mathbf{R}, \mathbf{I}, \mathbf{A})$, with a set of concepts **C**, relations **R**, individuals **I**, and axioms **A**, conforming to $(\mathcal{I}_{[0,t]}, \mathcal{H}_{[0,t]}) \overset{\eta}{\Leftrightarrow} \mathcal{O}_{[0,t]}$. Knowledge conceptual drift occurs at the timestamp $[t+1]$ to time stamp $[t+m]$ (m can be any number), when $\mathcal{O}_{[0,1]} \neq \mathcal{O}_{[t+1,\infty]}$, due to new concept introduced by a stakeholder, where the relation $(\mathcal{I}_{[0,t]}, \mathcal{H}_{[0,t]}) \overset{\eta}{\Leftrightarrow} \mathcal{O}_{[0,t]}$ changes to $(\mathcal{I}_{[t+1,t+m]}, \mathcal{H}_{[t+1,t+m]}) \overset{\Delta\eta}{\Leftrightarrow} \mathcal{O}_{[t+1,t+m]}$, as a result of real-world evolution.*

We use $\Delta\eta$ to denote knowledge conceptual drift between modeling layers (metamodel and ontology). This drift arises when ontology constructs become outdated due to domain changes, stakeholder needs, or model updates, leading to cross-layer inconsistencies driven by technical conceptual drift.

The threshold Φ (Definition 3.1) governs data-driven drift, such as label, feature, and concept drift, guiding maintainers on when to propagate updates based on drift severity, domain, and precision. Other drift types rely on maintainer judgment. Real-world contextualization of semantic drift types in a mobility use case is provided in [7].

4. RQ1: Semantic drift management requirements

This section outlines our mobility use case and elicits the requirements for DT frameworks, which will serve as the foundation for establishing metrics to assess how effectively DT platforms address semantic drift.



(a) Primary Data Model

(b) Evolved Data Model

Fig. 2. Data model - administration des transports publics.

4.1. Mobility use case

We provide an overview of the real-time mobility data model followed by some prototypical scenarios to assess the flexibility of DT frameworks for managing semantic drift.

4.1.1. Mobility network analysis

In this use case, we use the dataset of the administration des transports publics (ATP) in Luxembourg, provided by the *Mobilitéit API*.[1] This dataset provides information on buses, trams, train routes, and live traffic conditions. It is sourced from the *Luxembourg open data portal*[2] and is updated frequently to provide the most accurate information. The ATP dataset contains information on 2785 stops, has a size of 11.6 GB with 2 million objects, and is updated every 10 min. A data model representing the structure of data is given in Fig. 2(a). This data model directly reflects the structure of the data sourced from *Luxembourg open data portal* and more precisely the board data ,[3] which has no route explicitly defined. Route-like information is derived from departures and transport lines. Entities like `Arrival` and `Departure` are recorded separately, each linked to `Stops` and `Transport` types (`Bus`, `Tram`, `Train`).

4.1.2. Prototypical scenarios

The primary operator of the *Mobilitéit application* [4] is *Luxembourg Ministry*. In this sub-section, we present a set of prototypical scenarios

[1] https://data.public.lu/en/datasets/api-mobiliteit-lu/
[2] https://data.public.lu/en/
[3] https://cdt.hafas.de/opendata/apiserver/departureBoard?wadl
[4] https://www.mobiliteit.lu/en/

Table 2
Overview of prototypical scenarios.

Name	Semantic Drift Variant	Target Abstraction Layer	Changes
Scenario 4.1: *Bus Route*	Data Drift	Model Layer	Data value needs to be changed
Scenario 4.2: *Bus Arrival Time*	Structural Drift	Metamodel and Ontology Layer	New attributes need to be added
Scenario 4.3: *Transition in Distance Units*	Structural Drift	Model Layer	Value of *DistanceUnits* attribute needs to be changed
Scenario 4.4: *Temperature Sensor*	Technical Conceptual Drift	All Abstraction Layers	New entity needs to be added
Scenario 4.5: *Electric Buses*	Knowledge Conceptual Drift	All Abstraction Layers	New concept needs to be added

to reflect the changes introduced by the *Luxembourg Ministry* to transportation modes and the infrastructure. Further, we use these scenarios to elicit requirements and assess the flexibility of DT platforms.

Luxembourg Ministry can change information about the bus route, due to infrastructure changes, to reflect the real-world environment. This is illustrated by the Scenario 4.1, enabling us to study data-driven drift related to changes in data values.

Scenario 4.1. ***(Bus Route)*** *Suppose that the Ministry decides to change a specific bus route due to maintenance, repair, or construction along the route. This information can be updated by altering the line information of a specific bus route, as illustrated in Fig. 2. This change is limited to the model layer, as only the data values change.*

Luxembourg Ministry provides enhanced passenger experience, improved accuracy, operational efficiency, and data-driven decision-making through the *Mobilitéit application*. Predicted bus arrival times were introduced at each stop, as illustrated in Scenario 4.2, which is used to study structural drift.

Scenario 4.2. ***(Bus Arrival Time)*** *The Ministry used regression to predict the bus arrival time $\mathcal{Y}$ as $\mathcal{Y} = \mathbf{m}\mathcal{X} + \mathbf{b} + \epsilon$, where $\mathcal{X}$ is the distance (in kilometers),* $\mathbf{m}$ *is the slope indicating the rate at which the arrival time changes with distance,* $\mathbf{b}$ *is the intercept representing the baseline arrival time, and ϵ is the error term. This demands updates to metamodel and ontology layer, requiring new attributes to be incorporated into the bus entity.*

After some duration, the Ministry switched the measurement unit, due to a change in data collection technology as illustrated in Scenario 4.3.

Scenario 4.3. ***(Transition in Distance Units)*** *The unit shift of distance measurement from kilometers to meters affects the learning model and requires the slope* $\mathbf{m}$ *to be adjusted for consistency. This change is only limited to model layer.*

The addition of components in the physical environment is a significant process, as it provides new functionalities to the existing infrastructure and operating devices. Scenario 4.4 is used to study technical conceptual drift, which may affect metamodel and ontology.

Scenario 4.4. ***(Temperature Sensor)*** *The Ministry equips buses with temperature sensors to monitor overcrowding, recording values with units. This update impacts the model and necessitates a new temperature entity in the metamodel and ontology layer.*

Transportation modes are significant because they directly impact overall quality of life, shaping the economy, society, and environment. Evolving needs and challenges in society, ranging from environmental to technological concerns, can be addressed through the introduction of new transportation modes. We provide Scenario 4.5 to study knowledge conceptual drift.

Scenario 4.5. ***(Electric Buses)*** *The Ministry introduces electric buses to combat climate change, reduce emissions and noise. This shift impacts the metamodel, requiring stop entities to include charging hubs, and is reflected in the ontology layer first, influencing both model and metamodel layers.*

We provide an overview of our prototypical scenarios in Table 2. In order to use these scenarios to assess the flexibility of DT platforms, this ATP data model must be extended. We present our extended data model in Fig. 2(b), which corresponds to Scenarios 4.2–4.5. Scenario 4.1, requires only a change in data values and not in the structure of the data.

4.2. Requirement elicitation

The essential requirements, capabilities, and prospects of DT frameworks are given in [38], instantiating *ISO23247 standards* [39]. In this sub-section, we address **RQ1** by stating and extending requirements from [38], which are significant for addressing semantic drift in DT frameworks. We base our proposal on the various definitions of semantic drift introduced in Section 3 and detailed in [7].

4.2.1. [REQ 1] Basic synchronization

DT frameworks should enable two-way data exchange, supporting digital shadows, models or full twins to maintain synchronization with physical counterparts. Basic synchronization ensures that $\mathcal{T}$ aligns with $\mathcal{S}$ to prevent semantic drift (Definition 3.1). Applied to our use case, corresponding to Scenario 4.4, the `Bus` DT processes temperature sensor data and triggers an alarm when thresholds are exceeded.

4.2.2. [REQ 2] Data integration & management

A DT framework must enable data acquisition from IoT devices, scalable storage, and data fusion, while preserving historical and temporal data to mitigate semantic drift.

1. *Historical Data:* Preserving historical data ensures consistency as entities evolve. Historical data is significant in the context of Definition 3.2. In our use case (Scenario 4.4), integrating `TempSensor` into the `Bus` DT enhances environmental monitoring while requiring alignment with older models to prevent inaccurate analysis of operational and temperature-related data.
2. *Temporal Data:* In DT, time-stamped data preserves both current and historical states, crucial for addressing data-driven drift, as data values may change as the environment evolves. In our use case (Fig. 2), data related to entities like `Departures` and `Arrival` changes over time, enabling analysis of route adjustments or regulations on system efficiency.

4.2.3. [REQ 3] Modeling proficiencies

Modeling approaches are essential for detecting semantic drift, as they provide methods to identify changes across abstraction layers that impact DT performance [40]. A robust DT framework must offer a standardized way to express models, metamodels, and ontologies, capturing entity complexity with concepts like composition, aggregation, and generalization, while integrating model versioning to track and preserve the interpretability of historical data amidst structural changes. Modeling approaches are crucial for defining entities like `TempSensor` and `ElectricBus` (Definitions 3.2 and 3.3), requiring concepts such as aggregation and composition. As shown in Fig. 2, dynamic evolution in transport systems, including new entities like `ChargingPoint`, highlights the need for model versioning to manage outdated and current data structures. DT frameworks must support tools for defining and reusing ontologies to ensure data harmonization and seamless integration of new concepts, maintaining adaptability in evolving environments.

4.2.4. [REQ 4] Data validation

The framework must validate data against a schema/metamodel to ensure structural and semantic accuracy, and to ensure consistency between model and metamodel layer. In our use case (Fig. 2), adding attributes like `DistanceCovered` and `DistanceUnits` for bus arrival predictions requires metamodel verification to maintain model integrity.

4.2.5. *[REQ 5] Modifiability*

This refers to the ongoing ability to update DT models, metamodels, and ontologies, whenever their physical counterparts are modified. Modifiability is significant in order for any variant of semantic drift to be addressed, illustrated in Section 3. Applied to our use case, this corresponds to Scenario. 4.3, i.e., replacing data collection technology to measure distance. The new data collection technology will report distance values by using a different physical unit. This requirement can also be reflected in Scenario. 4.4 and 4.5 when a new sensor is added to a `Bus` to report temperature, and `ElectricBus` is introduced to replace fuel-based `Bus`. This implies modifying the existing model, metamodel, and ontologies in the DT framework. Additional requirements include convergence, real-time behavior, automation, interoperability, integration, security, provisioning, and re-usability, as detailed in [38].

5. RQ2: Evaluation metrics for digital twin framework

In this section, we propose metrics derived from requirements to assess the flexibility of language and data-specific DT frameworks in addressing semantic drift. The proposed metrics are formulated based on insights gathered from an analysis of the key contribution made in the field of DT [38,41,42], which evaluate capabilities like bi-directional integration, interoperability, intelligence, analytics, and community support. We also introduce a tiered compliance system for these metrics, adapted from [41] and *ISO 23247:2021–Automation systems and integration DT framework for manufacturing* .[5] It ranges from non-compliant (Tier 1) to fully compliant (Tier 4 or maximum level), to facilitate framework comparison in managing semantic drift.

5.1. *[M1] Communication*

We derive this metric from *[REQ 1]* and follow the categorization of DT integration levels as proposed in [43].

- *Tier 1:* None
- *Tier 2:* Supports Digital Model
- *Tier 3:* Supports Digital Shadow
- *Tier 4:* Supports Full DT

5.2. *[M2] Data acquisition, storage, and integration*

We derive this metric from *[REQ 2]*. A DT framework must capture data from IoT devices, provide scalable storage for large datasets in various formats, methods, and locations, and integrate data from diverse sources for a comprehensive view.

- *Tier 1:* No support for data acquisition, storage, and integration from IoT devices for maintaining historical and temporal data.
- *Tier 2:* Partial data acquisition, storage and integration from IoT devices for maintaining historical and temporal data.
- *Tier 3:* Full data acquisition, scalable storage, and basic integration from multiple IoT devices for maintaining historical and temporal data.

5.3. *[M3] Entity modeling*

This metric, derived from *[REQ 3]*, evaluates whether a DT framework offers a standardized approach for defining, maintaining, and versioning models, metamodels, and ontologies. It assesses the framework's ability to support essential modeling concepts, i.e., aggregation, composition, and generalization, ensuring comprehensive system representation. Furthermore, the validation ensures that the framework maintains records of both current and historical versions of entities and their attributes, preserving the integrity and interpretability of data despite changes over time due to real-world evolution.

- *Tier 1:* Provides a basic data structure to represent entities corresponding to IoT devices and may allow model versioning to support the coexistence of different versions of both the model and its metamodel.
- *Tier 2:* Provides a standardized language to model real-world entities with all essential concepts, and may allow model versioning to support the coexistence of different versions of both the model and its metamodel.
- *Tier 3:* Allows a standardized method for defining ontologies through web ontology language (OWL) or RDF, for semantic consistency in DT. It should also allow ontology versioning, to support the coexistence and maintenance of different versions of both ontology and its associated metamodel.

5.4. *[M4] Data conformance*

We derive this metric from *[REQ 4]*. A DT framework should validate data against its schema/metamodel to ensure accuracy, consistency, and alignment with predefined structural and semantic rules.

- *Tier 1:* Framework does not provide any verification or validation of data against schema/metamodel.
- *Tier 2:* Invalid data is detected by the framework.

5.5. *[M5] Semantic drift identification and characterization*

We derive this metric from *[REQ 5]*. We evaluate whether a DT framework provides a method or procedure to identify and characterize semantic drift. The *identification* of semantic drift implies the detection of changes and elements impacted by change, while *characterization* refers to a type of change that has occurred over the course of time.

- *Tier 1:* No methods for semantic drift identification and characterization.
- *Tier 2:* One or more methods to identify semantic drift.
- *Tier 3:* One or more methods to characterize semantic drift.

5.6. *[M6] Semantic drift support*

We derive this metric from *[REQ 5]*. We validate whether a DT framework provides a method or procedure to support the management of semantic drift against real-world evolution, while ensuring metamodel modifications without any disruption.

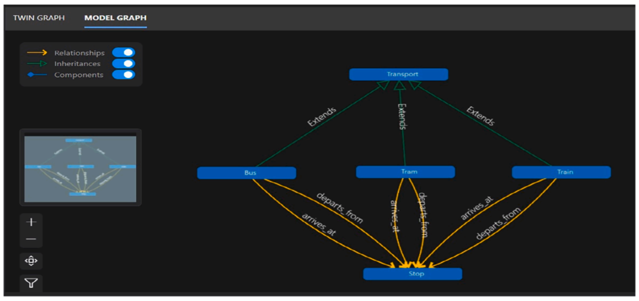
- *Tier 1:* No way to address any kind of semantic drift.
- *Tier 2:* Partial support, as methods are available for managing data-driven drift, i.e., label, feature and concept.
- *Tier 3:* Full support, as methods are available for managing structural, technical, and knowledge conceptual drift, provided that data is validated against the schema/metamodel.
- *Tier 4:* One or more methods to automatically update changes across different abstraction layers depicted in Fig. 1.

There are many other metrics given in [41] that can be used to assess the capabilities of the DT framework, but we limit our research to those metrics that are significant for assessing the flexibility of the DT to address semantic drift.

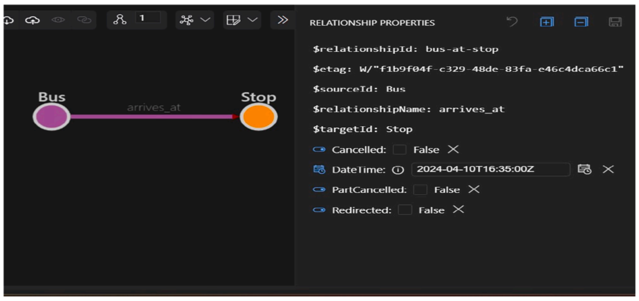
[5] https://www.iso.org/standard/78743.html

Table 3
Common vocabulary.

	Language-specific		Data-specific	
Modeling Layer	Azure DT	GreyCat	Fiware NGSI-LD	Eclipse Basyx
Ontology	–	–	–	–
Metamodel	Model Graph	Type Structure	Schema, Context Structure	Assets & Submodel
Model	Twin Graph	Node Definition	Context Data	Context Data



(a) Model Graph



```
RELATIONSHIP PROPERTIES
$relationshipId: bus-at-stop
$etag: W/"f1b9f04f-c329-48de-83fa-e46c4dca66c1"
$sourceId: Bus
$relationshipName: arrives_at
$targetId: Stop
Cancelled: False
DateTime: 2024-04-10T16:35:00Z
PartCancelled: False
Redirected: False
```

(b) Twin Graph

Fig. 3. DT graph-based representation.

6. Digital twin modeling

This section examines key capabilities of DT frameworks through the mobility use case, providing insights into their implementation and offering a common vocabulary (Table 3) for terms across different abstraction layers (Fig. 1).

6.1. *Language-specific digital twin framework*

This kind of DT framework framework uses a shared language structure tightly integrated with a programming language or DSL, requiring users to follow specific syntax and semantics. It provides specialized tools, libraries, and application programming interfaces (APIs) to enhance performance and usability for users familiar with the language. However, such frameworks may face challenges in flexibility and interoperability with external systems, require a steeper learning curve, and incur higher maintenance costs.

6.1.1. *Azure DT*

This IoT framework enables DT creation using a java-script object notation (JSON)-based language ,[6] i.e., DT definition language (DTDL), defining entities through components like telemetry, properties, and relationships. It employs a graph-based approach to create a twin graph and a model graph, representing real-world entities, requiring a backend for dynamic environments.

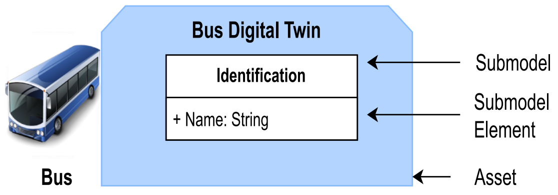


Fig. 4. Eclipse Basyx Bus Asset Representation.

The platform integrates seamlessly with services for data ingestion, event-driven workflows, and advanced analytics, offering standardized entity behavior definitions. represents DT corresponding to the mobility use case on the Azure DT framework. We modeled a *twin graph* and a *model graph* corresponding to the ATP data model using python-SDK, shown in Fig. 2(a) in Azure DT explorer. A glimpse of the bus entity DTDL for *model* and *twin graph* is given in Listing 1 and Listing 2, while Fig. 3 represents graph-based representation on Azure DT platform.

6.1.2. *GreyCat*

GreyCat [7] is offered by DataThings. It is a dynamic, evolving graph-based platform that integrates large time series and geographical data. It features an imperative executable language for defining types and manipulating graph nodes, combining graphs and time series into a scalable storage system. GreyCat organizes dynamic, unstructured data into multi-dimensional models known as *Many-Worlds Graphs*.

[6] https://azure.github.io/opendigitaltwins-dtdl/DTDL/v3/DTDL.v3.html

[7] https://datathings.com/

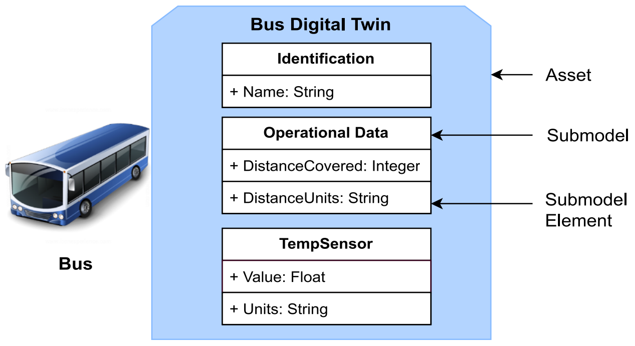


Fig. 5. Bus Asset with the Sensor in the Eclipse Baysx.

It supports real-time data processing, scalability, and flexible applications like live monitoring, data analytics, and DT solutions.The platform enables DT creation by leveraging historical data for predictions and simulations, using learning models to continuously refine and ensure real-time DT accuracy [44]. We modeled DT with typed structure and node definition provided by GreyCat, corresponding to the data model given in Fig. 2(a). Types are significant units in the GreyCat data structure, which defines aggregated typed fields into named types, later instantiated as an object, similar to the concept of classes in object-oriented programming languages. Nodes are also another significant unit, as GreyCat is an implementation of a graph structure, and the node constitutes the elements of the graph.

We provide a glimpse into our typed structure and node definition, focusing on the bus entity in Listings 3 and 4.

6.2. *Data-specific digital twin framework*

This category encompasses systems that manage DTs through well-defined data structures, ensuring communication between physical objects and their digital counterparts. These frameworks prioritize structured data, often utilizing universal formats like JSON, extensible markup language (XML) and RDF, rather than being tied to a specific programming language. By focusing on data organization, validation, and synchronization, they offer greater flexibility and interoperability across different platforms, leveraging standardized formats for efficient data exchange and system integration.

6.2.1. *Fiware NGSI-LD*

European telecommunications standards institute (ETSI) and the industry specification group on context information management (ISG CIM) developed the Fiware [8] next generation service interfaces – linked data (NGSI-LD) [45] specification as an extension of NGSI-V2, incorporating linked data principles to enhance data model richness and interoperability. NGSI-LD, which uses JSON-LD for context representation, is designed to manage context information essential for creating DTs of physical entities. By utilizing JSON-LD, NGSI-LD facilitates integration of heterogeneous data from diverse sources while preserving semantics. Scorpio, an open-source context broker offered by NGSI-LD, supports the management, access, and discovery of context data, modeling it in a graph structure with entities, attributes, and relationships. It also provides comprehensive methods, such as create, read, update, delete and advanced querying and subscription capabilities for interacting with DTs.

We modeled a DT that corresponded to the data model in Fig. 2(a), using NGSI-LD with scorpio context broker. We illustrate how this was used in Listing 5 to create an entity in the data model with a curl command to push the context structure and context data into the context broker. It should be noted that NGSI-LD does not offer any way to model abstract entities, composition, and the addition of attributes for a specific relation. The only way to represent abstract entities and composition is through an “**is_a**” and “**has**” relation.

6.2.2. *Eclipse Basyx*

This framework supports Java/.NET implementations of the asset administration shell (AAS) standard, providing infrastructure for AAS and registry servers via docker containers or JAR executables [46]. The AAS specification organizes assets, components, capabilities, and relationships through a hierarchical structure of submodels and properties. It includes features for creating, updating, and managing AAS objects, with a graphical user interface (GUI) client supporting OWL, RDF, JSON, or XML formats. As a comprehensive software development kit (SDK), it enables integration across various protocols such as transmission control protocol (TCP) and hypertext transfer protocol (HTTP) via the virtual automation bus (VAB), ensuring semantic interactions with AAS components. The platform supports secure HTTP connections and allows asset interfaces to trigger operations .[9]

We modeled the DT corresponding to the data model in Fig. 2(a) as a `Bus` AAS according to the context data, the shell that contains a `Bus` asset. At the same time, this can be represented by just one submodel, the `Identification submodel`, including different submodel elements, such as `name`, which is a property recognizing the bus.

We illustrate the representation of the `Bus AAS` in Listing 6. The `Bus AAS` illustrated in Listing 6 and Fig. 4, can be represented in XML, as shown in Listing 7.

7. RQ3: Semantic drift support in digital twin frameworks

This section addresses **RQ3** by presenting experiments that evaluate DT frameworks’ adaptability in managing semantic drift, using the proposed metrics to assess their effectiveness in maintaining semantic coherence across modeling layers and dynamic environments.

7.1. *Language-specific digital twin framework validation*

We consider Azure DT and GreyCat as language-specific DT frameworks and provide experiments in the context of prototypical scenarios to validate whether the framework is capable of addressing semantic drift or not.

7.1.1. *Azure DT*

This sub-section validates the Azure DT framework’s ability to manage diverse semantic drift scenarios, illustrated in sub-section. 4.1.2, leveraging DTDL’s graph-based modeling of heterogeneous models and metamodels with adaptive yet constrained updates.

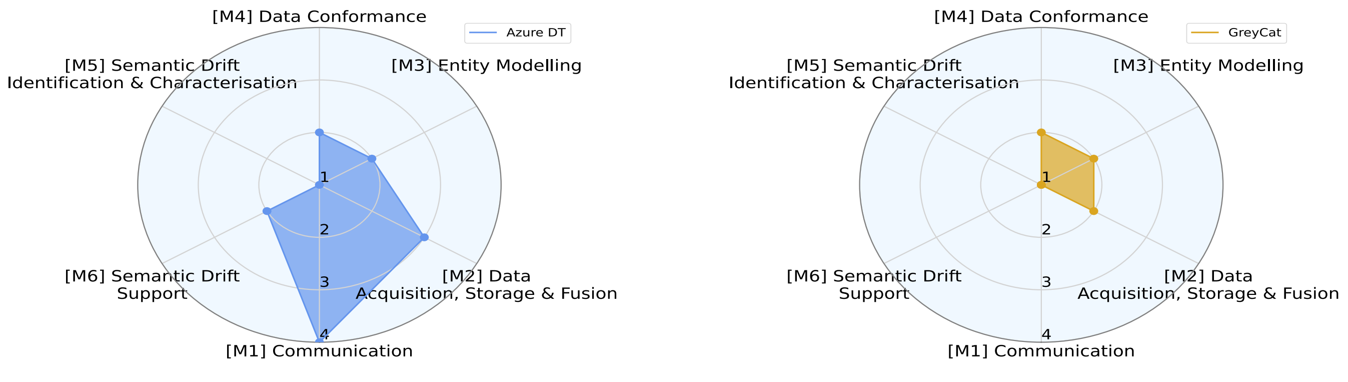
We validate Scenario 4.1 to address data-driven drift (label, feature and concept). We only need to update the *twin graph* existing in Azure DT, shown in Fig. 3(b), as it captures only the real-world data values. Azure DT provides support for addressing data-driven drift through a set of methods, i.e., **get_digital_twin ( )** and **update_digital_twin ( )**.

In the context of the given data model in Fig. 2 and Scenario 4.1, whenever the *Luxembourg Ministry* changes the line of a specific bus, which might be due to ongoing maintenance and repairs along the routes, relationship attributes and data values corresponding to other entities, i.e., `Stop` in our case, are also affected and need to be updated. A glimpse of this update can be observed in Listing 8.

Structural drift can be categorized under data-driven drift, which should be addressed in a timely manner. We need to validate Scenario 4.2, to assess whether or not we can address this kind of drift on Azure DT. Structural drift arises whenever there is a change in the structure of data. In the context of Scenario 4.2, when *Luxembourg Ministry* tries to calculate/predict the `bus arrival time` at a specific stop, taking into

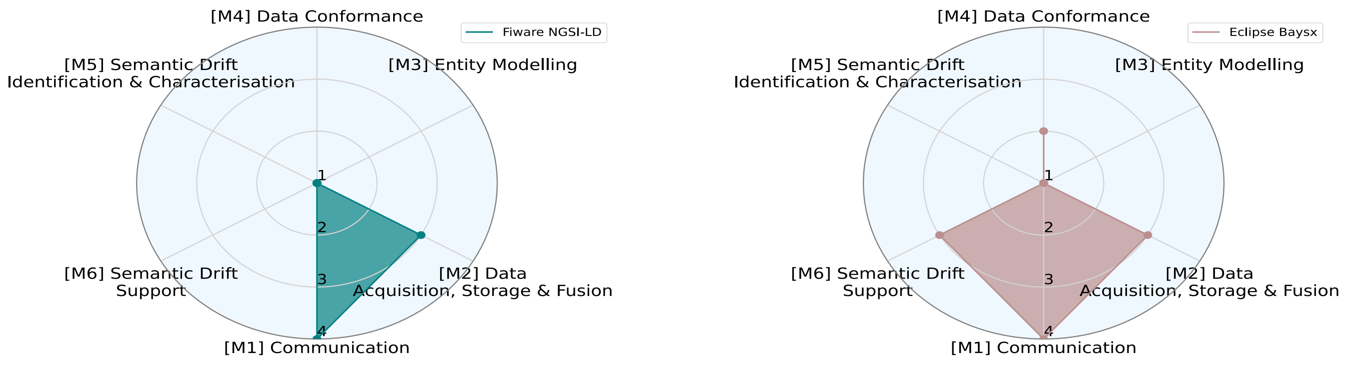
[8] https://fiware-datamodels.readthedocs.io/en/stable/ngsi-ld_howto/

[9] https://projects.eclipse.org/projects/dt.basyx



(a) Azure DT   (b) GreyCat

Fig. 6. Language Specific DT Framework Comparison.



(a) Fiware NGSI-LD   (b) Eclipse Basyx

Fig. 7. Data-Specific DT Framework Comparison.

```
url = os.getenv('AZURE_URL')
credential = DefaultAzureCredential()
client = DigitalTwinsClient(url, credential)
busId="Bus"
busTwin={
    "$metadata": {
    "$model": "dtmi:example:Bus;1"
  },
  "$dtId":busId,
  "Name": busId,
  "Operator":"RGTR",
  "Line":650,
}
created_twin = client.upsert_digital_twin(busId,busTwin)
```

Listing 1. Entity in Twin Graph(Azure DT).

consideration the distance covered, this involves adding new attributes to the model graph. The addition of new attributes, first on the *model graph* and then on the *twin graph*, is necessary to reflect the real-world environment. Listing 9 shows a glimpse of our failed experiment to update the *model graph*.

We observe from Listing 9 that Azure DT provides limited access to the *model graph*. We cannot access all the attributes and relationships of a single entity. Further, Azure DT does not provide any method, i.e., **update_model ( )**, to add new attributes or relations to the existing entities on the *model graph*. Although Azure DT provides methods to update the *twin graph*, this is only useful when the *model graph* is successfully updated to reflect real-world settings when new attributes are added. Validating Scenario 4.3 is only useful when we can successfully validate Scenario 4.2. We can make the same observation for Scenario 4.4 illustrating technical conceptual drift, in the event that the ministry decides to add a new sensor to the bus. This requires the addition of a new component on the *model graph* followed by its addition on the *twin graph*. Azure DT does not provide any support to address technical conceptual drift due to the limitations of methods to update the *model graph*.

Knowledge conceptual drift is another important semantic drift variant, which is illustrated by Scenario 4.5, in which the *Luxembourg Ministry* introduced `ElectricBus` to combat climate change. This concept should first be reflected on the ontology layer, and then on the meta-model layer (*model graph*) and model layer (*twin graph*).

The Azure DT framework does not offer a standardized way to create and manage ontologies, whereas DTDL can be integrated with or converted to RDF, to represent ontologies using a back-end. While we can add this new `ElectricBus` concept in the form of an entity on a meta-model layer (*model graph*), Listing 10 illustrates that we can add a new entity of `ElectricBus` extending from `Bus`, on the *model graph*, followed by the creation of a node representing `ElectricBus` on the *twin graph*.

At the same time, the creation of `ElectricBus` requires the creation of a `ChargingPoint` at stops, a change not allowed by Azure DT explorer, as it does not offer a method to update the *model graph* to add a new relation with existing entities. Complementary information in the context of the core metrics given in Section 5 for analyzing the DT framework, is provided as follows:

1. *[M1] Communication: Tier 1*, if the back-end is not considered. *Tier 4*, if the back-end is considered.
2. *[M2] Data Acquisition, Storage and Fusion: Tier 1* if back-end is not available, while Tier 3 if back-ends is available, enabling real-time data acquisition via Azure IoT Hub, storage in data lake, cosmos

```
url = os.getenv('AZURE_URL')
credential = DefaultAzureCredential()
client = DigitalTwinsClient(url, credential)
bus = { "@id": "dtmi:example:Bus;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2",
    "displayName": "Bus",
    "extends": ["dtmi:example:Transport;1"],
    "contents": [
    {
      "@type": "Property",
      "name": "Name",
      "schema": "string"
    },
    {
      "@type": "Relationship",
      "name": "arrives_at",
      "target": "dtmi:example:Stop;1",
      "properties": [
        {
          "@type": "Property",
          "name": "DateTime",
          "schema": "dateTime"
        }, # Adding other relationship attributes in similar way
      ]
    },
    {
      "@type": "Relationship",
      "name": "departs_from",
      "target": "dtmi:example:Stop;1",
      "properties": [
        {
          "@type": "Property",
          "name": "Scheduled_DateTime",
          "schema": "dateTime"
        }, # Adding other relationship attributes in similar way
      ]
    }
  ]
}
models = client.create_models([bus])
```

Listing 2. Entity in Model Graph(Azure DT).

```
abstract type Transport {
    operator: string;
    line: int;
}
type Bus extends Transport {
    name: String;
    arrival: Date;
    partcancelled: nodeTime<Status>?;
    cancelled: nodeTime<Status>?;
    redirected: nodeTime<Status>?;
    departure: Date;
    direction: String;
}
enum Status{
    TRUE;
    FALSE;
}
abstract type StatusUtil{
    static fn parse(val: String): Status{
        if (val == "true") {
            return Status::TRUE;
        } else {
            return Status::FALSE;
        }
    }
}
```

Listing 3. Bus Typed Structure.

```
var bus_by_name: nodeIndex<String, node<Bus>>;
fn main() {
    if (bus_by_name == null) {
        bus_by_name = nodeIndex<String, node<Bus>>::new(); }
    var reader = JsonReader::new("data.json");
    if (reader !=null ){
        while(reader.available() > 0) {
            var jsonArray = reader.read() as Array;
            for(positionInArray, vObject in  jsonArray) {
                var v_name = vObject.get("name") as String;
                var v_Node = bus_by_name.get(v_name);
                if (v_Node==null){
                     v_Node = node<Bus>::new(Bus{
                     name: v_name,
                     operator: vObject.get("operator") as String,
                     direction: vObject.get("direction") as String,
                     line: vObject.get("line") as int,
                     partCancelled: nodeTime<Status>::new(),
                     cancelled: nodeTime<Status>::new(),
                     redirected: nodeTime<Status>::new(),
                     arrival: Date::new( ),
                     departure: Date::new( ),
                });  }
            bus_by_name.set( v_name, v_Node);
            var iNode = *v_Node;
            var cancelled, partcancelled, redirected:
            cancelled=StatusUtil::parse(vObject.get("cancelled") as
      String);
            iNode.cancelled.set(cancelled);
            partcancelled=StatusUtil::parse(vObject.get("
      partcancelled") as String);
            iNode.partcancelled.set(partcancelled);
            redirected=StatusUtil::parse(vObject.get("redirected")
      as String);
            iNode.redirected.set(redirected);
            var arrival=Date::parse(vObject.get("arrivalDateTime")
      as String);
            iNode.arrival=arrival;
            var departure=Date::parse(vObject.get("departureDateTime
      ") as String);
            iNode.departure=departure;}}}
```

Listing 4. Bus Node Definition.

database, and blob storage, with data fusion through graph modeling and synapse analytics.

3. *[M3] Entity Modeling: Tier 2*, as metamodel is defined using DTDL, where version identifiers (e.g., `dtmi:example;1`) distinguish entities in the *twin graph* and *model graph*. Updated versions are created by incrementing the version number, ensuring access to both old and new models and metamodels.
4. *[M4] Data Conformance: Tier 2*, as invalid data is detected while creating a *twin graph*.
5. *[Metric 5] Semantic Drift Identification and Characterization: Tier 1* as there are no methods to identify any kind of semantic drift automatically.
6. *[M6] Semantic Drift Support: Tier 2*, as it provides methods to update twin graphs for addressing label, feature, and concept drift.

In conclusion, while Azure DT allows manual updates to the *model graph*, it lacks the capability to propagate these changes to the *twin graph*, hindering consistency across modeling layers.

7.1.2. GreyCat

DataThings specializes in DT solutions, through GreyCat's concept of scalable *Many-Worlds Graphs*, which integrate graphs and time series into multi-dimensional models for dynamic data management, as exemplified in Listings 3 and 4.

In this sub-section, we validate whether GreyCat is capable of handling different variants of semantic drift. We need to validate Scenario

```
{
    "id": "bus:001",
    "type": "Bus",
    "name": {
        "type": "String",
        "value": 650
    },
    "arrives":{
        "type":"Relationship",
        "object":"stop:001"
    },
    "departs":{
        "type":"Relationship",
        "object":"stop:001"
    },
    "is_a":{
        "type":"Relationship",
        "object":"transport:001"
    },
    "@context": [{"Bus": "urn:type:bus",
    "name": "unique_uri:name",
    "arrives": "unique_uri:arrives",
    "is_a": "unique_uri:is_a",
    "departs": "unique_uri:departs"},
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.
      jsonld"]
}
# Curl Query to Add Entity into Context Broker
$headers = @{
    "Content-Type" = "application/ld+json"
 }
Invoke-WebRequest -Uri 'http://localhost:9090/ngsi-ld/v1/entities' -
      Method Post -Headers $headers -Body (Get-Content Bus.jsonld -
      Raw)
```

Listing 5. Entity Creation(Bus.jsonld).

```
from basyx.aas import model
bus_asset_information = model.AssetInformation(
    asset_kind=model.AssetKind.INSTANCE,
    global_asset_id='http://acplt.org/Bus'
)
identifier_bus_aas = 'https://acplt.org/BusAAS'
bus_aas = model.AssetAdministrationShell(
    id_=identifier_bus_aas,
    asset_information=bus_asset_information
)
identifier_bus_submodel = 'https://acplt.org/identification'
bus_submodel = model.Submodel(
    id_=identifier_bus_submodel
)
semantic_reference_line = model.ExternalReference(
    (model.Key(
        type_=model.KeyTypes.GLOBAL_REFERENCE,
        value='http://acplt.org/Properties/BusName'
    ),)
)
name_property = model.Property(
    id_short='Name',
    value_type=model.datatypes.String,
    value="Bus-650",
    semantic_id=semantic_reference_line
)
bus_submodel.submodel_element.add(name_property)
```

Listing 6. Bus Asset.

```
from basyx.aas.adapter.xml import write_aas_xml_file
data_store: model.DictObjectStore[model.Identifiable] = model.
      DictObjectStore()
data_store.add(bus_submodel)

write_aas_xml_file(file='BusAAS.xml', data=data_store)
```

Listing 7. Bus Asset XML Representation.

4.1, to address data-driven drift (label, feature and concept). GreyCat is not capable of providing support, i.e., methods at runtime, to address any kind of semantic drift, i.e., data-driven, technical, and knowledge conceptual drift. Although in the event of technical and knowledge conceptual drift (Scenario 4.4 and 4.5), we can add abstraction and composition in a typed structure, it is only done manually, and not via a specific method, i.e., **update_type ( )** or **update_node ( )**, at runtime. Complementary information in the context of the core metrics given in Section 5 for analyzing the DT framework is given as follows:

1. *[M1] Communication: Tier 1*, as it only provides a way to model DT.
2. *[M2] Data Acquisition, Storage and Fusion: Tier 2*, as it provides data integration and management through the concept of *Many-World Graphs*.
3. *[M3] Entity Modeling: Tier 2*, as it uses the *GreyCat* typed structure and node definition.
4. *[M4] Data Conformance: Tier 2*, as it can detect invalid data against the typed structure and node definition.
5. *[M5] Semantic Drift Identification and Characterization: Tier 1*, as no support is provided.
6. *[M6] Semantic Drift Support: Tier 1*, as no methods are provided to support any kind of semantic drift management.

In summary, GreyCat provides foundational data integration and modeling support but lacks capabilities for ontology creation and addressing semantic drift.

7.2. Data specific digital twin framework validation

We consider Fiware NGSI-LD and Eclipse Basyx as data-specific frameworks and evaluate their capability to address semantic drift against prototypical scenarios.

7.2.1. Fiware NGSI-LD

We mainly assess the scorpio context broker for any kind of support for semantic drift. In the context of the properties outlined in sub-Section 5, NGSI-LD focuses on managing context information, representing the state of various entities over time.

We need to validate Scenario 4.1 to address data-driven drift (label, feature and concept). NGSI-LD is capable of updating the data values coupled with the schema/metamodel. A view of our updates for attributes of the `Bus` entity is provided in Listing 11. To update the data values of the `Stop` entity, we need to fetch relations associated with the `Bus` entity using the curl command, shown in Listing 12, followed by an update of the attributes of the `Stop` entity, similar to that shown in Listing 11.

We need to validate Scenario 4.2, to address structural drift, which is a type of data-driven drift. Structural drift should be addressed in a timely manner, to maintain consistency in DT. The addition of new attributes in the existing context structure and the context data is illustrated in Listing 13. After the addition of attributes, if we need to change the units of measurement to record distance values as illustrated in Scenario 4.3, we will use the same curl command as in Listing 13, with the only modifying attribute value being `DistanceUnits`.

Executing these commands with the Scorpio context broker instance successfully adds new attributes or modifies existing attributes to the

```
twin=client.get_digital_twin(bus_id)
patch = [
    {
        "op": "replace",
        "path": "/Line",
        "value": 551
    }
]
client.update_digital_twin(bus_id, patch)
relationships = client.list_relationships(bus_id)
target_id=""
related_twin_ids = set()
for relationship in relationships:
       target_id=relationship['$targetId']
patch = [
    {
        "op": "replace",
        "path": "/Name",
        "value": "Foetz, Am Brill"
    }, # Replacing all attributes in a similar way
]
client.update_digital_twin(target_id, patch)
```

Listing 8. Twin Graph Updation.

```
url = os.getenv('AZURE_URL')
credential = DefaultAzureCredential()
client = DigitalTwinsClient(url, credential)
model=client.get_model("dtmi:example:Bus;1")
print(model) #Returns the metadata and definition
```

Listing 9. Accessing Model Graph.

```
electricbus= {
  "@id": "dtmi:example:ElectricBus;1",
  "@type": "Interface",
  "displayName": "ElectricBus",
  "@context": "dtmi:dtdl:context;2",
   "extends": ["dtmi:example:Bus;1"],
   "contents": [
    {
        "@type": "Property",
        "name": "BatteryCapacity",
        "schema": "integer"
    },] }
models = client.create_models([electricbus])
```

Listing 10. Extending Model Graph.

bus entity to reflect real-world changes. It should be noted that NGSI-LD updates the context structure along with the context data, which is how a data-specific DT works, and provides no validation of whether or not the new or existing data conforms to the metamodel.

Technical conceptual drift is another important semantic drift variant, which invokes changes in the metamodel/schema. We validate Scenario 4.4 to address technical conceptual drift. We provide an overview of our experiment in Listing 14 and 15, in which we added a new component to the `Bus` entity, and then a relation. It should be noted that NGSI-LD does not provide a standardized way to model composition, and we therefore added a new entity in the context broker and linked it with the `Bus` entity through a **has** relation.

Similarly to address knowledge conceptual drift, which arises as a result of a change in the ontology layer, if a new concept such as an `ElectricBus` is added by the *Luxembourg Ministry* to the existing data model, this change is first reflected in the ontology, then in the metamodel layer (context structure) and model layer (context data) (Scenario 4.5).

```
$body = @{
        line = @{
                type = "Number"
                value = 551
        }
    } | ConvertTo-Json

Invoke-RestMethod -Uri 'http://localhost:1026/v2/entities/urn:ngsi-
      ld:transport:001/attrs' -Method Post -ContentType 'application
      /json' -Body $body
```

Listing 11. Update Attributes (Fiware NGSI-LD).

```
curl 'http://localhost:1026/v2/entities/urn:ngsi-ld:bus:001?attrs=
      arrives' -Method GET
```

Listing 12. Get Relation (Fiware NGSI-LD).

NGSI-LD uses the same method to add new entities to the context data, as shown in Listing 14, and links them to the bus entity through an **is_a** relation, as shown in Listing 15, as NGSI-LD does not provide a standardized way to model an **is_a** relation (aggregation). The addition of a new concept, i.e., `ElectricBus`, is limited to the metamodel layer (context structure) as NGSI-LD does not provide support for defining and maintaining ontologies.

NGSI-LD introduces a range of challenges in data modeling and validation, with the most significant being its flexible approach to data-schema coupling. This lack of a rigid schema or metamodel hinders automatic validation of key elements such as required fields or data types, creating a potential for inconsistencies. The absence of predefined constraints, such as limits on value ranges or string lengths, leaves room for erroneous data to be stored, undermining the integrity and uniformity of the DT models. This flexibility, while offering adaptability, imposes additional complexity on developers, who must resort to external tools to enforce data validation and ensure schema compliance. Complementary information about the core metrics for analyzing the DT framework discussed in Section 5 is as follows:

1. *[M1] Communication: Tier 4*, as it supports a bi-directional flow of data and services through a feedback loop.
2. *[M2] Data Acquisition, Storage & Management: Tier 3*, as it enables the seamless integration of diverse data sources with real-time updates through data connectors and a context broker, transforming formats like JSON and XML to NGSI-LD.
3. *[M3] Entity Modeling: Tier 1*, as it uses an entity attribute relationship (EAR) model for DT modeling, without providing a standardized method for modeling essential concepts, i.e., aggregation, generalization, composition, and model versioning.
4. *[M4] Data Conformance: Tier 1*, as it does not support any verification and validation of data against the schema.
5. *[M5] Semantic Drift Identification and Characterization: Tier 1*, as it provides no methods to identify and characterize semantic drift.
6. *[M6] Semantic Drift Support: Tier 1* as it provides methods to update context information but does not provide data validations, which is necessary for managing drift.

7.2.2. *Eclipse Basyx*

When working with Eclipse Basyx, the approach follows the AAS concept, and is therefore mostly oriented towards the manufacturing industry. We model the DT as illustrated in sub-Section 6.2.2, corresponding to our mobility data model shown in Fig. 2(a). In the context of the metrics outlined in sub-Section 5, we validate Scenario 4.1, to address data-driven drift (label, feature and concept). Eclipse Basyx is capable of updating the context data, which is coupled with assets and submodels. Eclipse Basyx does not provide a standard way to model abstraction. We therefore link the `Transport` and `Bus` assets through an

```
$body = @{
        distanceCovered = @{
                type = "Number"
                value = 500
        },
        distanceUnits = @{
                type = "String"
                value = km
        },
     } | ConvertTo-Json

Invoke-RestMethod -Uri 'http://localhost:1026/v2/entities/urn:ngsi-
      ld:bus:001/attrs' -Method Post -ContentType 'application/json'
       -Body $body
```

Listing 13. Add Attributes (Fiware NGSI-LD).

```
{
    "id": "TempSensor:001",
    "type": "TempSensor",
    "value": {
        "type": "Number",
        "value": 30
    },
    "units": {
        "type": "String",
        "value": "Celcius"
    },
    "@context": [{"Sensor": "urn:type:TempSensor",
    "value": "unique_uri:name",
    "units": "unique_uri:units",
     },
    "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.
      jsonld"]
}
# Curl Query to Add Entity into Context Broker
$headers = @{
    "Content-Type" = "application/ld+json"
 }
Invoke-WebRequest -Uri 'http://localhost:9090/ngsi-ld/v1/entities' -
      Method Post -Headers $headers -Body (Get-Content Sensor.jsonld
       -Raw)
```

Listing 14. Entity Creation in Fiware NGSI-LD (Sensor.jsonld).

```
$body = @{
          has = @{
                type = "Relationship"
                value = "urn:ngsi-ld:TempSensor:001"
          }
     } | ConvertTo-Json

Invoke-RestMethod -Uri 'http://localhost:1026/v2/entities/urn:ngsi-
      ld:bus:001/attrs' -Method Post -ContentType 'application/json'
       -Body $body
```

Listing 15. Add Relation (Fiware NGSI-LD).

```
relationship_element = model.RelationshipElement(
    id_short="is_a",
    first=model.ModelReference.from_referable(transport_aas),
    second=model.ModelReference.from_referable(bus_aas)
)
bus_submodel.submodel_element.add(relationship_element)
```

Listing 16. Add Relation (Eclipse Basyx).

Table 4
Comparative analysis of DT frameworks.

		Language-specific		Data-specific	
Metrics	Tiers	Azure DT	GreyCat	Fiware NGSI-LD	Eclipse Basyx
[M1]	*Tier 1*		✓		
	Tier 2				
	Tier 3				
	Tier 4	✓		✓	✓
[M2]	*Tier 1*				
	Tier 2		✓		
	Tier 3	✓		✓	✓
[M3]	*Tier 1*			✓	✓
	Tier 2	✓	✓		
	Tier 3				
[M4]	*Tier 1*			✓	
	Tier 2	✓	✓		✓
[M5]	*Tier 1*	✓	✓	✓	✓
	Tier 2				
	Tier 3				
[M6]	*Tier 1*		✓	✓	
	Tier 2	✓			
	Tier 3				✓
	Tier 4				

is_a relation, as well as providing a way to address data-driven drift (label, feature and concept) by updating the value of the `line` attribute in the `transport` asset.

The addition of an **is_a** relation between the `Transport` and `Bus` assets is illustrated in Listing 16, whereas Listing 17, shows our update of the attribute values of the `Transport` asset.

We validate Scenario 4.2 for structural drift. We need to add a new `submodel`, named `operational data`, to the `Bus` asset to predict the `bus arrival time`. This can be seen in the `submodel`, i.e., `operational data`, in the `Bus` asset provided in Listing 18. For Scenario 4.3, we followed the same procedure as described in Listing 17 to switch units of measurements, i.e. from kilometers to meters, changing the value of `DistanceUnits`.

We validate Scenario 4.4 for technical conceptual drift. Since Eclipse Basyx does not provide a standard way of modeling composition, we needed to add a temperature sensor in the form of a `submodel` to the `Bus` asset. Similarly, we added a *submodel* for `operational data` as depicted in Listing 18.

We validate Scenario 4.5 for knowledge conceptual drift. The `ElectricBus` concept must first be reflected on the ontology layer, then on the metamodel layer (assets, submodels), and finally on the model layer (context data). According to our observations and experiments, Eclipse Basyx does not provide a standard way of defining and maintaining ontologies. It also lacks a proper way of expressing abstraction, which needs to be added when a new `ElectricBus` concept is introduced by the Ministry to replace the traditional buses on metamodel layer (assets, submodels).

Overall, we can summarize that Eclipse Basyx provides only the concept of a submodel, with which we can model different concepts, i.e., composition. A submodel concept may provide support to address data-driven, structural, and technical conceptual drift as Eclipse Basyx provides methods to add and update the submodel. An illustration of the `Bus` asset along with essential submodels is provided in Fig. 5. The complementary information relating to core metrics, given in Section 5 for analyzing the DT framework, is as follows:

1. *[M1] Communication: Tier 4*, as it supports full DT.
2. *[M2] Data Acquisition, Storage and Fusion: Tier 3*, as it provides multiple protocol support through OPC UA, MQTT, and HTTP REST. It can manage data from both edge devices and cloud systems. It also provides data storage and persistence through SQL database (PostgreSQL) for registering assets and No-SQL database (MongoDB) for storing events and data.
3. *[M3] Entity Modeling: Tier 1*, as it uses AAS implementation at the metadata level, and does not provide ways to model abstraction and composition, whereas the concept of submodel can be used to model

```
for element in transport_submodel.submodel_element:
    if isinstance(element, model.Property) and element.id_short == '
      Line':
        element.value = new_line_value
        break
data_store: model.DictObjectStore[model.Identifiable] = model.
      DictObjectStore()
data_store.add(transport_submodel)
write_aas_xml_file(file='TransportAAS.xml', data=data_store)
```

Listing 17. Updating Attribute Value (Eclipse Basyx).

```
operational_bus_submodel = 'https://acplt.org/operational_data'
operational_data_submodel = model.Submodel(
    id_=operational_bus_submodel
)
semantic_reference_distanceCovered = model.ExternalReference(
    (model.Key(
        type_=model.KeyTypes.GLOBAL_REFERENCE,
        value='http://acplt.org/Properties/DistanceCovered'
    ),)
)
distance_Covered = model.Property(
    id_short='Distance Covered',
    value_type=model.datatypes.Integer,
    value="500",
    semantic_id=semantic_reference_distanceCovered
)
semantic_reference_distanceUnits = model.ExternalReference(
    (model.Key(
        type_=model.KeyTypes.GLOBAL_REFERENCE,
        value='http://acplt.org/Properties/DistanceUnits'
    ),)
)
distance_Units = model.Property(
    id_short='Distance Units',
    value_type=model.datatypes.String,
    value="km",
    semantic_id=semantic_reference_distanceUnits
)
operational_data_submodel.submodel_element.add(distance_Covered)
operational_data_submodel.submodel_element.add(distance_Units)
```

Listing 18. Adding Submodel (Eclipse Basyx).

composition to a limited extent. It also provides model versioning through AAS as an asset can have multiple versions, enabling users to manage and track evolved assets.

4. *[M4] Data Conformance: Tier 2*, as invalid data can be detected against a schema/metamodel.
5. *[Metric 5] Semantic Drift Identification and Characterization: Tier 1*, as it provides no method or procedure for identifying and characterizing semantic drift.
6. *[M6] Semantic Drift Support: Tier 3*, as it provides methods to add and update submodels.

7.3. Results

We present a graphical analysis of key attributes for semantic drift in DT frameworks, using evaluation metrics from Section 5, ranked on a 1 to 4 tier scale. Fig. 6 and Fig. 7 show spider diagrams of language- and data-specific DT frameworks, offering a quantitative view of results and tool coverage for semantic drift. This is complemented by both qualitative and quantitative assessments summarized in Table 4. Note that for Azure DT, we provide a spider diagram when the back-end is fully available. We can summarize through these spider diagrams that no DT framework provides a method or procedure for identifying and characterizing semantic drift, however, two of the DT frameworks, i.e., Azure DT and Eclipse Basyx, provide partial support to address semantic drift variants.

8. Discussion: Digital twin framework

This section provides qualitative discussion on studied frameworks. We identify key challenges, benefits, and limitations of language and data-specific DT frameworks in addressing semantic drift. The DT frameworks examined exhibit diverse features and functionalities tailored to various industrial domains, making the selection of the appropriate framework crucial for successful DT adoption. In our case, the mobility use case was compatible with all frameworks implemented; however, each presented unique drawbacks in managing and addressing semantic drift, as discussed in Section 7. Our findings highlight that while no DT framework currently fully addresses the identification and characterization of semantic drift, this presents a valuable opportunity for further innovation and development in the field. Below, we summarize our observations regarding semantic drift support across DT frameworks:

1. **Eclipse Basyx** offers the most robust support for addressing and managing semantic drift among the platforms evaluated. It excels in handling structural and technical conceptual drift by enabling updates to submodels through its robust implementation of the AAS standard. Additionally, it supports the conversion of assets into RDF and XML, thereby aiding standardization efforts. However, improvements are needed in managing knowledge conceptual drift as some modeling concepts, i.e., generalization, are missing, and there is limited support for defining and maintaining ontologies.
2. **Azure DT** is a powerful tool, particularly when supported by a robust back-end, and it offers a unified schema/metamodel for defining DT structures, enhancing compatibility with commercial solutions. It addresses data-driven drift effectively through updates to the *twin graph*. While its support for technical conceptual drift is limited due to the absence of mechanisms for updating the *model graph*, Azure DT allows the conversion of DTDL into RDF, laying a foundation for semantic standardization. With further enhancement, it has a strong potential for broader semantic drift management.
3. **Fiware NGSI-LD** offers effective solutions for managing context information and supports flexible updates, however, its flexible data and schema coupling features pose challenges for data validation and drift management, whereas its versatility presents opportunities for improvement in the handling of semantic drift variants. Additionally, it does not provide standardized methods for defining and maintaining ontologies, which are essential for tackling knowledge conceptual drift. However, it does facilitate the reuse of existing ontologies through URI, which may provide a pathway for managing knowledge conceptual drift.
4. **GreyCat**, although not currently equipped to manage semantic drift, shows promise for future development. As the platform evolves, the incorporation of mechanisms for handling semantic drift and ontologies could significantly enhance its value in managing complex semantic scenarios.

We recognize the existence of various other DT frameworks as listed in [41], both commercial and open-source, that merit further exploration. Our study solely focuses on language and data-specific frameworks, addressing **RQ1**, **RQ2** and **RQ3**. This is because effective semantic drift analysis requires frameworks supporting at least three critical layers: data, model, and metamodel. We also exclude some tools which do not provide data collection facilities. We further emphasize that addressing semantic drift is a shared responsibility between DT frameworks and modeling experts. While frameworks must enable robust mechanisms for model migration, versioning, traceability, and automated runtime updates, their efficacy ultimately depends on expert-driven modeling practices to ensure adaptive and reliable DT evolution and maintenance. Additionally, managing horizontal drift, detailed in [7], which arises from misaligned peer models across systems or domains at the same abstraction layer, is crucial for maintaining interoperability and integrity in distributed DT ecosystems. It would be beneficial if such management were supported directly by DT frameworks through enhanced coordination, synchronization, and semantic governance mechanisms. Furthermore, Our evaluation revealed that DT frameworks offer limited support for detecting malicious data. Therefore, it is imperative they incorporate scalable semantic drift management for high-volume data streams, complemented by secure mechanisms ensuring data integrity, access control, and model provenance.

9. Conclusion

DTs hold transformative potential by mirroring real-world behaviors through models, metamodels, and ontologies, enhancing systems-of-systems with deep insights and optimization. However, dynamic real-world induces semantic drift, necessitating proactive strategies to preserve DT effectiveness over time. Our research contributes to assess the effectiveness of DT frameworks in *identifying, characterizing, and providing any level of support* for managing semantic drift, addressing **RQ1**, **RQ2** and **RQ3**. We provide an elicitation of requirements for managing semantic drift, informed by a practical mobility use case following a set of prototypical scenarios. Further, we draw on a set of evaluation metrics, from the elicited requirements, to perform a robust evaluation of language and data-specific DT frameworks for highlighting their practical value in managing semantic drift.

Our findings show that existing DT frameworks lack clear procedures for identifying or characterizing semantic drift and offer only limited support for addressing various drift types. This study focuses on evaluating current frameworks without implementing or empirically validating potential solutions, which remains a key direction for future work.

Looking ahead, we aim to develop flexible and robust solutions for managing semantic drift across multiple modeling layers, considering the behavior and execution aspects of DT. These solutions will integrate model management capabilities to support model synchronization and evolution, ultimately enhancing the reliability, adaptability, and relevance of DT in dynamic environments.

CRediT authorship contribution statement

Faima Abbasi: Writing – original draft, Visualization, Validation, Methodology, Data curation, Conceptualization; **Cédric Pruski:** Validation, Supervision, Methodology, Investigation, Conceptualization; **Jean-Sébastien Sottet:** Validation, Supervision, Funding acquisition, Data curation, Conceptualization.

Data availability

The data which is used in this research is already available on Luxembourg open data portal (link given in research)

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

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

References

[1] S. Abburu, A.J. Berre, M. Jacoby, D. Roman, L. Stojanovic, N. Stojanovic, Cognitive digital twins for the process industry, in: Proceedings of the the Twelfth International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 2020), Nice, France, 2020, pp. 25–29.

[2] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, A.Y.C. Nee, Enabling technologies and tools for digital twin, J. Manuf. Syst. 58 (2021) 3–21.

[3] A.L. Suárez-Cetrulo, D. Quintana, A. Cervantes, A survey on machine learning for recurring concept drifting data streams, Expert Syst. Appl. 213 (2023) 118934.

[4] E. Whiting, S. Andrews, Drift and erosion in software architecture: summary and prevention strategies, in: Proceedings of the 2020 the 4th International Conference on Information System and Data Mining, 2020, pp. 132–138.

[5] J. Chen, F. Lecue, J.Z. Pan, S. Deng, H. Chen, Knowledge graph embeddings for dealing with concept drift in machine learning, J. Web Semant. 67 (2021) 100625.

[6] R. Jongeling, J. Fredriksson, F. Ciccozzi, A. Cicchetti, J. Carlson, Towards consistency checking between a system model and its implementation, in: Systems Modelling and Management: First International Conference, ICSMM 2020, Bergen, Norway, June 25–26, 2020, Proceedings 1, Springer, 2020, pp. 30–39.

[7] 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, 2024, pp. 419–430.

[8] E.B. Ouedraogo, A. Hawbani, X. Wang, Z. Liu, L. Zhao, M.A.A. Al-qaness, S.H. Alsamhi, Digital twin data management: a comprehensive review, IEEE Trans. Big Data (2025), 11, pp. 2224–2243.

[9] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: a review, IEEE Trans. Knowl. Data Eng. 31 (12) (2018) 2346–2363.

[10] A. Armijo, D. Zamora-Sánchez, Integration of railway bridge structural health monitoring into the internet of things with a digital twin: a case study, Sensors 24 (7) (2024) 2115.

[11] M.N. Hasan, S.U. Jan, I. Koo, Wasserstein GAN-based digital twin-inspired model for early drift fault detection in wireless sensor networks, IEEE Sens. J. 23 (12) (2023) 13327–13339.

[12] R. Gupta, B. Tian, Y. Wang, K. Nahrstedt, TWIN-ADAPT: continuous learning for digital twin-enabled online anomaly classification in IoT-driven smart labs, Future Internet 16 (7) (2024) 239.

[13] Y. Wu, L. Liu, Y. Yu, G. Chen, J. Hu, An adaptive ensemble framework for addressing concept drift in IoT data streams, Authorea Preprints (2023).

[14] J. Michael, L. Cleophas, S. Zschaler, T. Clark, B. Combemale, T. Godfrey, D.E. Khelladi, V. Kulkarni, D. Lehner, B. Rumpe, et al., Model-driven engineering for digital twins: opportunities and challenges, Syst. Eng. (2025, 659–670, 28).

[15] M. Kannapinn, M. Schäfer, O. Weeger, TwinLab: a framework for data-efficient training of non-intrusive reduced-order models for digital twins, Eng. Comput. (2024).

[16] J. Mertens, S. Klikovits, F. Bordeleau, J. Denil, Ø. Haugen, Continuous evolution of digital twins using the dartwin notation, Softw. Syst. Model. (2024), 24, 1405–1426.

[17] W. Sun, S. Lei, L. Wang, Z. Liu, Y. Zhang, Adaptive federated learning and digital twin for industrial internet of things, IEEE Trans. Ind. Inf. 17 (8) (2020) 5605–5614.

[18] 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), IEEE, 2021, pp. 1–4.

[19] H.M. Muctadir, L. Cleophas, M. van den Brand, 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), IEEE, 2024, pp. 152–159.

[20] H.M. Muctadir, E. Kamburjan, L. Cleophas, M. van den Brand, A consistency management framework for digital twin models, Available at SSRN 5105174.

[21] T. Kühne, Multi-dimensional multi-level modeling, Softw. Syst. Model. 21 (2) (2022) 543–559.

[22] J. Liu, Q. Ji, H. Zhou, C. Du, X. Liu, M. Li, A multi-dimensional evolution modeling method for digital twin process model, Robot. Comput. Integr. Manuf. 86 (2024) 102667.

[23] F. Salama, I. Sezgin, E. Korkan, S. Käbisch, S. Steinhorst, HoloWoT: a first step towards mixed reality digital twins for the industrial internet of things, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 318–321.

[24] 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), in: Design, Automation and Test in Europe Conference (DATE 2025), 2025.

[25] E. Karabulut, S.F. Pileggi, P. Groth, V. Degeler, Ontologies in digital twins: a systematic literature review, Future Gener. Comput. Syst. 153 (2024) 442–456.

[26] F. Abbasi, M. Muzammal, Q. Qu, F. Riaz, J. Ashraf, SNCA: semi-supervised node classification for evolving large attributed graphs, Big Data Min. Anal. 7 (3) (2024) 794–808.

[27] F. Abbasi, M. Muzammal, Q. Qu, A decentralized approach for negative link prediction in large graphs, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, 2018, pp. 144–150.

[28] M. Muzammal, F. Abbasi, Q. Qu, R. Talat, J. Fan, A decentralised approach for link inference in large signed graphs, Future Gener. Comput. Syst. 102 (2020) 827–837.

[29] J. Persson, S. Johannsen, The design of an ontology management tool (2024).

[30] Q. Bao, G. Zhao, Y. Yu, S. Dai, W. Wang, The ontology-based modeling and evolution of digital twin for assembly workshop, Int. J. Adv. Manuf. Technol. 117 (2021) 395–411.

[31] Z. Ren, J. Shi, M. Imran, Data evolution governance for ontology-based digital twin product lifecycle management, IEEE Trans. Ind. Inf. 19 (2) (2022) 1791–1802.

[32] F. Abbasi, M. Muzammal, K.N. Qureshi, I.T. Javed, T. Margaria, N. Crespi, Exploiting optimised communities in directed weighted graphs for link prediction, Online Soc. Netw. Media 31 (2022) 100222.

[33] L. Vogt, FAIR knowledge graphs with semantic units: a prototype, arXiv preprint arXiv:2311.04761 (2023).

[34] E. Kamburjan, N. Bencomo, S.L. Tapia Tarifa, E.B. Johnsen, Declarative lifecycle management in digital twins, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 353–363.

[35] G. Pronost, F. Mayer, M. Camargo, L. Dupont, Digital twins along the product lifecycle: a systematic literature review of applications in manufacturing: [version 2; peer review: 2 approved, 2 approved with reservations], Digit. Twin 1 (2) (2024) 3.

[36] E.B. Gulcan, F. Can, Unsupervised concept drift detection for multi-label data streams, Artif. Intell. Rev. 56 (3) (2023) 2401–2434.

[37] K. Wang, L. Zhang, Z. Jia, H. Cheng, H. Lu, J. Cui, A framework and method for equipment digital twin dynamic evolution based on IExATCN, J. Intell. Manuf. 35 (4) (2024) 1571–1583.

[38] 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 Softw. 39 (2) (2021) 53–61.

[39] I. ISO, DIS 23247-1 automation systems and integration–digital twin framework for manufacturing, Int. Organ. Stand. Geneva Switz. (2020).

[40] F. Steimann, J. Gößner, T. Mück, On the key role of composition in object-oriented modelling, in: International Conference on the Unified Modeling Language, Springer, 2003, pp. 106–120.

[41] S. Gil, P.H. Mikkelsen, C. Gomes, P.G. Larsen, Survey on open-source digital twin frameworks–A case study approach, Softw. Pract. Exper. 54 (6) (2024) 929–960.

[42] Y.K. Liu, S.K. Ong, A. Nee, State-of-the-art survey on digital twin implementations, Adv. Manuf. 10 (1) (2022) 1–23.

[43] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn, Digital twin in manufacturing: a categorical literature review and classification, IFAC PapersOnline 51 (11) (2018) 1016–1022.

[44] T. Hartmann, F. Fouquet, A. Moawad, R. Rouvoy, Y. Le Traon, GreyCat: efficient what-if analytics for data in motion at scale, Inf. Syst. 83 (2019) 101–117.

[45] U. Ahle, J.J. Hierro, FIWARE for data spaces, Designing Data Spaces, 395, 2022.

[46] K. Schweichhart, Reference architectural model industrie 4.0 (rami 4.0), An Introduction 40 (2016).